Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Mojca Lorber

Mere podobnosti nizov

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Jurij Mihelič

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Iskanje in primerjanje nizov je eden izmed ključnih računskih problemov, ki se pogosto pojavlja kot sestavni del drugih algoritmov. V okviru diplomske naloge se osredotočite na ugotavljanje podobnosti in različnosti nizov. Preglejte in definirajte obstoječe mere podobnosti ter s tem povezane računske probleme. Opišite algoritme za izračun izbranih mer podobnosti in podajte njihovo računsko zahtevnost. Pregled področja naj bo matematično obarvan, pri čemer kar se da poenotite izrazoslovje in notacijo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Mojca Lorber sem avtor diplomskega dela z naslovom:

Mere podobnosti nizov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom doc. dr. Jurija Miheliča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 26. januarja 2016 Podpis avtorja:

Rada bi se zahvalila za strokovno pomoč pri izdelavi diplomske naloge mentorju doc. dr. Juriju Miheliču. Zahvaljujem se Anji Košir, prof. slov., za lektoriranje ter mojim bližnjim za moralno podporo.

Kazalo

Seznam oznak

Množice

A, B	Množici A in B
$\|A\|$	Število različnih elementov v množici A
$A \subseteq B$	A je podmnožica od B
$A \cup B$	Unija množic A, B
$A \cap B$	Presek množic A, B
$A \setminus B$	Razlika množic A in B
A^C	Komplement množice A
$P(A)$	Potenčna množica množice A
(x, y)	Urejen par elementov x in y
$A \times B$	Kartezični produkt dveh množic A in B

Abecede in nizi

a, b, c	Znaki, simboli oz. črke a, b, c
s, t, u, v, w	Nizi oz. besede s, t, u, v in w
Σ	Abeceda
$\|s\|$	Dolžina niza s
ε	Prazen niz
$s[i]$	Znak na i-tem mestu niza s
Σ^n	Množica vseh nizov dolžine n nad abecedo Σ
Σ^*	Množica vseh možnih nizov nad abecedo Σ
Σ^+	Množica vseh nepraznih nizov nad abecedo Σ
L	Jezik nad Σ
st	Konkatenacija oz. produkt nizov s in t

$s \preceq_{fakt} t$	Niz s je podniz oz. faktor od t
q-gram	Podniz dolžine q
$s[i..j]$	Podniz $s[i]s[i+1]\ldots s[j]$ niza s
$s \sqsubset t$	Niz s je prefiks ali predpona od t
$s \sqsupset t$	Niz s je sufiks ali pripona od t

Grafi

$G = (V, E)$	Graf G, kjer je V množica vozlišč in E množica povezav
$u \sim v$	Sosednji vozlišči u, v
$e = \{u, v\}$	Dvosmerna povezava med u, v v neusmerjenemu grafu
$e = (u, v)$	Usmerjena povezava med u, v v digrafu
$w(e)$	Utež povezave e v uteženem grafu
$G' \subseteq G$	G' je podgraf od G
$S = (e_1, e_2, \ldots, e_k)$	Sprehod S v grafu G, dolžine k
$d(S)$	Dolžina sprehoda
$d_G(u, v)$	Razdalja v grafu G med vozliščema u, v; dolžina najkrajše poti od u do v

Primerjalke

$p(s, t)$	Primerjalka ali primerjalna funkcija p nizov s, t
$d(s, t)$	Prema primerjalka ali mera različnosti
$s(s, t)$	Obratna primerjalka ali mera podobnosti
$d \sim \phi$	Ekvivalentni primerjalki d, ϕ

Faktorizacija nizov

$d_{pref}(u, v)$	Razdalja predpone
$lcp(u, v)$	Dolžina najdaljše skupne predpone od u in v (angl. longest common prefix)
$d_{suff}(u, v)$	Razdalja pripone
$lcsuff(u, v)$	Dolžina najdaljše skupne pripone od u in v (angl. longest common suffix)
$d_{fact}(u, v)$	Razdalja podniza
$lcf(u, v)$	Dolžina najdaljšega skupnega podniza od u in v (angl. longest common factor)

Različnost in razdalja

σ	Skripta ali sled urejanja, ki predstavlja zaporedje operacij, ki transformirajo s v t (angl. edit script, trace)
$u \to v$	Osnovna operacija urejanja, ki pretvori u v v (angl. basic edit operation)
$\mathbb{B}$	Množica osnovnih operacij urejanja
$ED(s,t)$	Razdalja urejanja (angl. edit distance)
$Lev(s,t)$	Levenshteinova razdalja
$DamLev(s,t)$	Damerau-Levenshteinova razdalja
$Ham(s,t)$	Hammingova razdalja
$Indel(s,t)$	Razdalja podzaporedja ali Indel razdalja
$\delta(a \to b)$	Stroškovna funkcija δ, ki dodeli osnovnim operacijam urejanja iz $\mathbb{B}$ posamezne stroške
$Sub(a,b)$	Strošek zamenjave znakov a z b
$Del(a)$	Strošek brisanja znaka a
$Ins(b)$	Strošek vstavljanja znaka b
$cost(\sigma)$	Strošek sledi urejanja (angl. cost function)
$\mathcal{S}(s,t)$	Množica vseh sledi urejanja, ki pretvorijo s v t
$reED(s,t)$	Omejena razdalja urejanja (angl. restricted edit distance)
$Lcs(s,t)$	Množica vseh najdaljših skupnih podzaporedij od s,t
$lcs(s,t)$	Dolžina najdaljšega skupnega podzaporedja od s,t
$sim(s,t)$	Podobnost med nizoma s in t

Vizualizacija

$\alpha = (\tilde{s}, \tilde{t})$	Poravnava dveh nizov s in t; kjer velja $\tilde{s}[i] \to \tilde{t}[i] \in \mathbb{B}$
(a,b)	Poravnan par zankov a,b
$-$	Simbol, ki predstavlja presledek oz. luknjo v poravnavi
$cost(a,b)$	Strošek poravnanega para, ki je enak $Sub(a,b)$
$cost(a,-)$	Strošek poravnanega para, ki je enak $Del(a)$
$cost(-,b)$	Strošek poravnanega para, ki je enak $Ins(b)$
$cost(\alpha)$	Strošek poravnave α (angl. cost of alignment)
$align(s,t)$	Optimalna poravnava nizov s,t

$\mathcal{A}(s,t)$	Množica vseh poravnav nizov s,t
$G(s,t)$	Graf urejanja G nizov s,t
V	Množica vozlišč grafa $G(s,t)$
E	Množica usmerjenih povezav grafa $G(s,t)$
$label(e)$	Funkcija, ki povezavam $e \in E$ določi poravnane pare $(s[i],t[j])$
$cost(label(e))$	Strošek povezave $e \in E$ je enak vrednosti oznake
$Dot[m,n]$	Tabela Dot imenovana točkovni diagram (angl. dotplot)

Sorodni problemi

N-$Lcs(S)$	Množica vseh najdaljših skupnih podzaporedij N nizov iz S
N-$lcs(S)$	Dolžina najdaljšega skupnega podzaporedja N nizov iz S
N-$Scs(S)$	Množica vseh najkrajših skupnih super-zaporedij N nizov iz S
N-$scs(S)$	Dolžina najkrajšega skupnega super-zaporedja N nizov iz S
N-$Scns(S)$	Množica vseh najkrajših skupnih ne-podzaporedij N nizov iz S
N-$scns(S)$	Dolžina najkrajšega skupnega ne-podzaporedja N nizov iz S
$Hcs(s,t)$	Množica vseh najtežjih skupnih podzaporedij N nizov iz S
$hcs(s,t)$	Dolžina najtežjega skupnega podzaporedja N nizov iz S

Povzetek

Diplomska naloga *Mere podobnosti nizov* proučuje problem primerjanja nizov, kjer nas zanimajo ujemanja, ki dovoljujejo tudi napake. Takšnemu problemu pravimo tudi problem približnega ujemanja nizov in njegov bistveni del je definicija modela napak ter s tem izbira mere podobnosti oz. različnosti. V nalogi na začetku izvedemo splošen pregled mer, potem pa se v nadaljevanju osredotočimo na skupino mer, ki temelji na operacijah urejanja nizov. Definicija razdalje med nizoma je tako določena s stroškom operacij, ki prvi niz najbolj optimalno preuredi v drugega. V tem sklopu nato opišemo nekaj algoritmov na osnovi metode dinamičnega programiranja ter dodamo še par njihovih nadgradenj. S pomočjo primera nazorno prikažemo njihovo izvajanje ter z analizo predstavimo tudi njihove računske zahtevnosti.

Ključne besede: podobnost, različnost, mera podobnosti, primerjanje nizov, poravnava nizov, razdalja urejanja, najdaljše skupno podzaporedje.

Abstract

The thesis *String similarity measures* examines string matching problem, where we are interested in matchings allowing errors. Such problem is also called approximate string matching problem, and its essential part is the definition of error model and by this the type of a similarity or dissimilarity measure. In the beginning of the thesis we present a general overview of measures, then we further focus on the group of measures based on the edit operations on strings. The definition of such distance between strings is established with the cost of operations that are needed for an optimal transformation from one string to another. Further on, we describe a few algorithms based on dynamic programming, and then we add a couple of upgraded versions. With a help of an example we try to demonstrate their performance and analyse their computational complexity.

Keywords: similarity, dissimilarity, similarity measure, string matching, string alignment, edit distance, longest common subsequence.

Poglavje 1

Uvod

Ko govorimo o merah podobnosti nizov, pravzaprav govorimo o problemu primerjanja nizov, kadar so dovoljene tudi napake. Problem približnega ujemanja nizov se je prvič pojavil v šestdesetih oz. sedemdesetih letih prejšnjega stoletja. Največja motivacija je v tistih časih prihajala s področij bioinformatike, procesiranja signalov in preiskovanja besedil in še danes so to največja področja uporabe [Nav01].

Osrednji predmet proučevanja v bioinformatiki so genetska zaporedja DNK[1] in zaporedja proteinov, ki jih lahko predstavimo z dolgimi nizi, sestavljenimi iz znakov določene abecede (npr. A, C, G, T za DNK). Ugotavljanje podobnosti oz. različnosti genetskih zaporedij je pomembno pri preučevanju evolucijske zgodovine in pri odkrivanju funkcij genov (na primer ugotovitev povezave med genom, ki povzroča raka, in genom za rast). Pri tem je preiskovanje nizov z natančnim ujemanjem skoraj neuporabno. Meritve zaporedij namreč lahko vsebujejo različne vrste napak. Poleg tega obstajajo tudi razlike med zaporedji, ki nastanejo zaradi mutacij in evolucijskih sprememb. Te tipične napake oz. spremembe lahko definiramo kot operacije, ki so potrebne za transformacijo enega niza v drugega. Operacijam se glede na verjetnost dodelijo stroški in cilj predstavlja minimizacijo stroška. Aktual-

[1]DNK je kratica za deoksiribonukleinsko kislino in predstavlja molekulo, ki je nosilka genetske informacije v vseh živih organizmih.

nost tega področja nam med drugim dokazuje odmevnost projekta določitve celotnega človeškega genoma. To pomembno odkritje je podlaga za nadaljnje raziskovanje dednih bolezni in spoznavanje novih načinov zdravljenja le teh.

Druga motivacija je s področja procesiranja signalov. Sem spada prepoznavanje govora, kjer je treba na podlagi zvočnega signala prepoznati besedilno sporočilo. Že preprost primer določitve besede je zelo kompleksen, saj imamo lahko dodatne omejitve, kot je kompresiran signal, ali pa nepravilno izgovorjavo delov besedila. Tudi tu natančna ujemanja niso mogoča. Potem je tu problem popravljanja napak, ki nastanejo med prenosom signala. Fizični prenos signala je namreč nagnjen k nastanku napak in pravilen prenos lahko zagotovimo z možnostjo obnovitve prvotnega sporočila s pomočjo korekcijskih kod. Procesiranje signalov je zelo aktivno področje še danes. Hitro povečevanje multimedijskih podatkov kliče po rešitvah na področju vsebinskega preiskovanja slikovnih, zvočnih in video podatkov. Veliko priložnosti je v prihodnosti na področju nepisne komunikacije med človekom in računalnikom. In tudi zanesljive korekcijske kode bodo še naprej aktualne zaradi popularnosti brezžičnih tehnologij, saj je zrak prenosni medij nizke kakovosti.

Preiskovanje besedil oz. popravljanje napačno črkovanih besed v pisni obliki je zelo staro področje. In mere podobnosti nizov so tu zelo uporabne, saj lahko večino teh napak popravimo s samo enim vstavljanjem, brisanjem, zamenjavo ali transpozicijo znakov. Eden izmed bolj zahtevnih področij uporabe je iskanje ustreznih informacij v velikih tekstovnih zbirkah. Klasični načini primerjanja nizov niso več zadostni, saj tekstovne zbirke postajajo vse večje, bolj heterogene in nagnjene k napakam. Danes praktično ni več izdelka za preiskovanje besedila, ki ne bi dovoljeval vsaj delnih neujemanj. Poleg tega obstajajo še druge aplikacije za obdelavo besedila, kot so črkovalniki, vmesniki naravnega jezika itd.

Ostala področja uporabe so še: prepoznavanje ročne pisave, odkrivanje virusov in vdorov, kompresija slik, podatkovno rudarjenje, prepoznavanje vzorcev, optično prepoznavanje znakov, primerjava datotek, osveževanje za-

slonov ...

V diplomski nalogi so najprej v drugem poglavju predstavljeni matematični temelji s področja teorije množic. Nato bodo predstavljene definicije in lastnosti abecede in nizov, temu bo sledil kratek pregled iz teorije grafov. Najpomembnejši del iz poglavja osnovnih pojmov pa je razdelek primerjave nizov, kjer bodo definirani pojmi mer podobnosti.

Osnovam bo v tretjem poglavju sledil kratek pregled mer podobnosti oz. različnosti nizov. Za tem pa se bomo v naslednjem poglavju osredotočili samo na področje mer podobnosti na osnovi operacij urejanja. Najbolj poznana in pogosto uporabljena med njimi je razdalja urejanja (angl. edit distance). Natančno jo bomo definirali ter predstavili vse njene izpeljanke: Hammingova razdalja, Levenshteinova razdalja, Damerau-Levenshteinova razdalja, Indel razdalja ... Vse bomo podkrepili tudi s primeri algoritmov. Vmes bomo navedli različne metode vizualizacije primerjav, kot so optimalna poravnava, graf urejanja ter točkovni diagram. Določitev razdalje med dvema nizoma je tesno povezana tudi s problemom iskanja najdaljšega skupnega podzaporedja dveh nizov. Predstavljen bo tudi koncept podobnosti na primeru lokalne primerjave.

Na koncu naloge, v zadnjem poglavju, še navedemo sorodne probleme, ki se pojavljajo na področju primerjave nizov. Sledi samo še sklep.

Poglavje 2

Osnovni pojmi

Predstavimo množice, nize [HUM01], [SM97], [CHL01], [Str15], grafe [CLRS09] ter primerjanje nizov [Bre98] in [Bat85].

2.1 Množice

Definicija 1. Množica A *je končna ali neskončna zbirka različnih objektov, kjer vrstni red ni pomemben ter se večkratna pojavitev enakih objektov ignorira. Te objekte imenujemo* elementi x.

Glavni pojem teorije množic je relacija *pripadnosti*. Element x lahko pripada množici A ($x \in A$), ali pa ne ($x \notin A$).

Množico, ki ne vsebuje nobenega elementa, imenujemo *prazna možica*; označimo jo z $\emptyset$ ali $\{\}$.

Univerzalna množica U je množica vseh elementov, o katerih razpravljamo v okviru nekega izbranega problema.

Množica je lahko podana na:

- **intenzionalni način:** z naštevanjem elementov, recimo $A = \{x, y, z, \ldots\}$,
- **ekstenzionalni način:** z izjavno formulo, recimo $A = \{x;\ \phi(x)\}$ (pri tem je $\phi(x)$ izjavna formula, ki razen x ne vsebuje prostih nastopov spremenljivk).

Moč množice $|S|$ je število elementov, ki pripada množici.

Definicija 2. *Definirajmo enakost* (2.1)*, inkluzijo* (2.2) *in strogo inkluzijo* (2.3) *množic:*

$$A = B \iff \forall x : (x \in A \iff x \in B) \tag{2.1}$$

$$A \subseteq B \iff \forall x : (x \in A \Longrightarrow x \in B) \tag{2.2}$$

$$A \subset B \iff A \subseteq B \wedge A \neq B \tag{2.3}$$

Če velja $A \subseteq B$, rečemo, da je množica A *podmnožica* množice B. Če pa velja $A \subset B$, rečemo, da je množica A *prava podmnožica* množice B.

Trditev 1. *Množici* A *in* B *sta enaki, če vsebujeta iste elemente. To je res, samo če je množica* A *podmnožica množice* B*, hkrati pa je tudi množica* B *podmnožica množice* A*:*

$$A \ = \ B \iff (A \subseteq B) \wedge (B \subseteq A).$$

Osnovne operacije nad množicami so:

- **unija:** $(A \cup B) = \{x;\, x \in A \vee x \in B\}$,
- **presek:** $(A \cap B) = \{x;\, x \in A \wedge x \in B\}$,
- **razlika:** $(A \setminus B) = \{x;\, x \in A \wedge x \notin B\}$.

Definicija 3. Komplement *množice definiramo kot:*

$$A^C = U \setminus A,$$

kjer je U *univerzalna množica.*

Definicija 4. Potenčna množica *vsebuje vse možne podmnožice neke množice* A*:*

$$P(A) = \{X;\, X \subseteq A\}.$$

Dvoelementarna množica $\{x, y\}$ predstavlja neurejen par, tj. $\{x, y\} = \{y, x\}$. *Urejen par* (x, y) pa predstavlja zapis, sestavljen iz dveh elementov, pri čemer je pomembno, kateri element je na prvem in kateri na drugem mestu. Elementa x in y, ki nastopata v zapisu urejenega para, imenujemo komponenti para. Urejeni par (x, y) ni enak urejenemu paru (y, x).

Urejeno n-terico pa definiramo takole:

$$(x_1, x_2, x_3, \ldots, x_n) = (\ldots((x_1, x_2), x_3), \ldots, x_n).$$

Definicija 5. Kartezični produkt *dveh množic A in B je množica, sestavljena iz urejenih parov, ki imajo prvo komponento iz množice A in drugo iz množice B. Podamo jo na naslednji način:*

$$A \times B = \{(x, y);\ x \in A \wedge y \in B\}.$$

2.2 Abecede in nizi

Definicija 6. Abeceda *(angl. alphabet) je končna neprazna množica* znakov *ali* simbolov $a, b, c \ldots$ *(angl. characters or symbols)*[1]*. Označimo jo s* Σ.

Primeri abeced:
- binarna abeceda ($\Sigma = \{0, 1\}$)
- slovenska abeceda ($|\Sigma| = 25$)
- angleška abeceda ($|\Sigma| = 26$)
- DNK abeceda ($\Sigma = \{A, C, G, T\}$)
- abeceda aminokislin ($|\Sigma| = 20$)

Definicija 7. Niz *s (angl. string)*[2] *je končno urejeno zaporedje znakov ali simbolov iz abecede* Σ.

[1] Včasih se uporablja tudi izraz *črke* (angl. letters).
[2] Včasih poimenovan *beseda* (angl. word).

V nizih se lahko znaki ponavljajo. Na primer, 1011 in 001 sta niza iz binarne abecede $\Sigma = \{0, 1\}$.

Dolžina niza s, označeno $|s|$, je število položajev znakov v nizu[3], kjer je $|s| \in \mathbb{N}_0$. Niz dolžine 0 imenujemo *prazen niz* in ga označimo ε. Znak v nizu s na i-tem mestu označimo $s[i]$, kjer indeksi znakov zajemajo vrednosti $i \in \{1, \ldots, |s|\}$. Tako lahko zapišemo osnovno definicijo identitete med poljubnima dvema nizoma s in t:

Definicija 8.

$$s = t \iff |s| = |t| \ \wedge \ s[i] = t[i], \quad \forall i \in \{1, \ldots, |s|\}$$

Σ^n je množica vseh nizov dolžine n nad abecedo Σ.

Definicija 9.

$$\Sigma^n = \{s_1, s_2, \ldots, s_{|\Sigma|^n}\}, \quad \textit{kjer } |s_i| = n \ \wedge \ s_i[j] \in \Sigma,$$

za $\forall i \in \{1, \ldots, |\Sigma|^n\} \ \wedge \ \forall j \in \{1, \ldots, n\}$.

$\Sigma^0 = \{\varepsilon\}$, za vsako abecedo Σ. Na primer, če je $\Sigma = \{0, 1\}$, potem je $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$ in tako naprej.

S Σ^* označimo množico vseh možnih nizov končne dolžine, ki jih lahko tvorimo z znaki iz abecede Σ:

Definicija 10.

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots = \bigcup_{n \in \mathbb{N}_0} \Sigma^n$$

Na primer, $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

[3]Pogosto se dolžina niza opisuje kot število znakov v nizu, kar ni formalno pravilno. V nizu 1011 sta namreč 2 znaka, 0 in 1, položajev znakov pa je 4. V nadaljevanju ne bomo vedno formalno korektni in bomo kdaj izpustili izraz "položaj" z obzirom, da se bralec tega zaveda.

Včasih želimo iz množice nizov Σ^* izključiti prazen niz ε. Množica vseh nepraznih nizov nad abecedo Σ se označi Σ^+:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$
$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

Množica nizov, izbranih iz neke Σ^*, se imenuje *jezik* (angl. language). Če je Σ abeceda in $L \subseteq \Sigma^*$, potem je L jezik nad Σ. Zgled je slovenski jezik, kjer slovar slovenskega knjižnega jezika predstavlja množico nizov nad slovensko abecedo (25 črk). Še en primer, če je $\Sigma = \{0, 1\}$, je množica nizov s sodim številom ničel $\{\varepsilon, 1, 00, 11, 001, 010, 100, 111, 0000, 0011, 0101, 0110, \dots\}$ jezik nad Σ.

Konkatenacija oziroma *produkt* dveh nizov s in t je niz sestavljen iz zaporedja znakov iz s, ki jim sledi zaporedje znakov iz t. Označimo st ali pa tudi $s \cdot t$ kadar želimo poudariti mesto združitve nizov. Bolj natančno:

Definicija 11. *Dana sta niza s in $t \in \Sigma^*$. Če je niz s sestavljen iz i znakov: $s = s[1]s[2]\cdots s[i]$ in je niz t sestavljen iz j znakov: $t = t[1]t[2]\cdots t[j]$, $(i, j \in \mathbb{N}_0)$, potem* konkatenacija *st predstavlja niz dolžine $i+j$: $st = s[1]s[2]\cdot \cdots s[i]t[1]t[2]\cdots t[j]$.*

Konkatenacija nizov je asociativna operacija:

$$\forall\, s, t, w \in \Sigma^*\colon\ (st)w = s(tw) = stw, \tag{2.4}$$

ni pa komutativna:

$$\forall\, s, t \in \Sigma^*\colon\ st \neq ts. \tag{2.5}$$

Nevtralni element produkta je ε:

$$\forall\, w \in \Sigma^*\colon\ \varepsilon w = w\varepsilon = w. \tag{2.6}$$

Iz lastnosti (2.4) in (2.6) sledi, da množica Σ^* skupaj z operacijo konkatenacije tvori monoid.

Podajmo še zgled; naj bo $s = 1011$ in $t = 001$, potem je $st = 1011001$ in $ts = 0011011$.

Podzaporedje (angl. subsequence) niza s je zaporedje t, ki ga dobimo iz s z odstranitvijo $|s| - |t|$ položajev znakov.

Definicija 12. *Niz t je* podzaporedje *niza s, če obstaja $|t| + 1$ nizov w_0, $w_1, w_2 \ldots, w_{|t|} \in \Sigma^*$, tako da je $s = w_0 t[1] w_1 t[2] \ldots t[|t|] w_{|t|}$.*

Prazen niz ε je podzaporedje vsakega niza. Kadar je niz t podzaporedje niza s, rečemo, da je s *super-zaporedje* (angl. supersequence) od t.

Podniz (angl. substring) niza s je niz t sestavljen iz zaporednih znakov iz s, z ohranjenim vrstnim redom. Bolj formalno:

Definicija 13. *Niz t je* podniz *ali* faktor *niza s, če obstajata niza u in $v \in \Sigma^*$, tako da je $s = utv$.*

Posebna primera podnizov sta *prefiks* in *sufiks*. Če je $u = \varepsilon$, potem je t prefiks od s in če je $v = \varepsilon$, je t sufiks od s. Več o tem v nadaljevanju.

Relacija "je podniz od"(označimo jo "$\preceq_{fakt}$") nad množico Σ^* je delna urejenost, kar pomeni, da je:

$$\text{refleksivna: } \forall\, s \in \Sigma^*\colon\ s \preceq_{fakt} s, \tag{2.7}$$

$$\text{antisimetrična: } \forall\, s, t \in \Sigma^*\colon\ s \preceq_{fakt} t,\ t \preceq_{fakt} s \implies s = t, \tag{2.8}$$

$$\text{tranzitivna: } \forall\, s, t, w \in \Sigma^*\colon\ s \preceq_{fakt} t,\ t \preceq_{fakt} w \implies s \preceq_{fakt} w. \tag{2.9}$$

Podniz nekega niza s je tudi njegovo podzaporedje, vendar niso vsa podzaporedja niza s tudi njegovi podnizi. Podniz določajo strožja pravila kot podzaporedje, ki predstavlja posplošitev podniza. Kadar je niz t podniz niza s, rečemo, da je s *super-niz* (angl. superstring) od t.

Q-gram[4] je podniz fiksne dolžine q. Niz dolžine m ima lahko največ $m - q + 1$ različnih q-gramov.

Podniz w se včasih lahko pojavi večkrat znotraj niza u. V nekaterih primerih je pomembno, da znamo razbrati te pojavitve, za kar lahko uporabimo *intervale*.

[4]Znan tudi pod imenom n-gram oz. k-mer. Izraz k-mer je bolj s področja bioinformatike.

Definicija 14. Interval *niza* s *je množica zaporednih indeksov* $[i..j]$*, tako da* $1 \leq i \leq j+1 \leq |s|+1$.

Interval vsebuje vse indekse med i in j, vključno z i in j. Za določeni interval $[i..j]$ in niz s lahko definiramo niz $s[i..j]$, ki predstavlja podniz $s[i]s[i+1]\ldots s[j]$ niza s, pod pogojem da je $i \leq j$. Če je $i > j$ dobimo prezen niz ε. Zato obstaja za vsak podniz t niza s vsaj en interval $[i..j]$ niza s, tako da je $t = s[i..j]$.

Prefiks ali *predpona* niza s je podniz oblike $s[1..j]$, za $0 \leq j \leq |s|$.

Definicija 15. *Niz* t *je* predpona *od* s *natanko takrat, ko obstaja niz* $u \in \Sigma^*$*, tako da* $s = tu$*. Pri tem je* $|t| \leq |s|$.

Včasih se sklicujemo na predpono od s z natančno k znaki, $0 \leq k \leq |s|$, kar označimo kot $prefiks(s,k)$.

Sufiks ali *pripona* niza s je podniz oblike $s[i..|s|]$, za $1 \leq i \leq |s|+1$.

Definicija 16. *Niz* t *je* pripona *od* s *natanko takrat, ko obstaja niz* $u \in \Sigma^*$*, tako da* $s = ut$*. Velja* $|t| \leq |s|$.

Notacija $sufiks(s,k)$ opisuje pripono niza s z k število znakov, $0 \leq k \leq |s|$.

Ker sta prefiks in sufiks posebna primera podnizov, za njiju prav tako veljajo lastnosti delne urejenosti: refleksivnost (2.7), antisimetričnost (2.8) in tranzitativnost (2.9). Poleg tega za relaciji "je prefiks od"(oznaka $\sqsubset$) in "je sufiks od"(oznaka $\sqsupset$) veljata še dodatni lastnosti:

$$\forall\, s,t,u \in \Sigma^*\colon\ s \sqsubset t \ \wedge\ u \sqsubset t \implies s \sqsubset u \ \vee\ u \sqsubset s,$$
$$\forall\, s,t,u \in \Sigma^*\colon\ s \sqsupset t \ \wedge\ u \sqsupset t \implies s \sqsupset u \ \vee\ u \sqsupset s.$$

Prazen niz ε je predpona in pripona vsakega niza s. Za katerakoli niza t in s ter vsak znak a, je t sufiks od s, natanko takrat, ko je ta sufiks od sa in podobno za prefiks:

$$\forall\, s,t \in \Sigma^* \ \wedge\ \forall\, a \in \Sigma\colon\ t \sqsupset s \iff ta \sqsupset sa,$$
$$\forall\, s,t \in \Sigma^* \ \wedge\ \forall\, a \in \Sigma\colon\ t \sqsubset s \iff at \sqsubset as.$$

2.3 Grafi

Definicija 17. Graf *je urejen par* $G = (V, E)$*, kjer je* V *množica vozlišč in* $E \subseteq V \times V$ *množica povezav grafa* G*.*

Če obstaja povezava $e = \{u, v\} \in E$ pravimo, da sta vozlišči u in v *sosednji* v grafu G in pišemo $u \sim v$. Za povezavi pravimo, da sta *sosednji*, če imata kako skupno vozlišče.

Pravimo, da je graf *enostaven*, če nima niti zank (povezava, katere začetno vozlišče je tudi končno vozlišče) niti vzporednih povezav (več povezav, ki imajo skupno začetno in končno vozlišče).

Poznamo *neusmerjene* (angl. undirected graph) in *usmerjene* (angl. directed graph) grafe.

Definicija 18. Neusmerjen graf *je graf* G*, v katerem so povezave* E *dvosmerne. Dvosmerna povezava* $e = \{u, v\} \in E$ *je neurejen par vozlišč* $u, v \in V$*.*

To pomeni, da povezava $e = \{u, v\}$ nima usmeritve in se zapisa $\{u, v\}$ in $\{v, u\}$ ne razlikujeta.

Definicija 19. Usmerjen graf *ali* digraf *je graf* G*, katerega povezave* E *so usmerjene. Usmerjena povezava* $e = (u, v) \in E$ *predstavlja urejen par vozlišč* $u, v \in V$*, pri čemer je vozlišče* u *začetek ali izvor povezave* (u, v) *in vozlišče* v *konec ali ponor povezave* (u, v)*.*

Zapis $e = (u, v)$ torej določa, da povezava e poteka od vozlišča u proti vozlišču v.

Za prikaz grafov lahko uporabimo tudi matriko sosednosti:

$$A(G) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad a_{ij} = \begin{cases} 1, & \text{če } v_i \sim v_j \\ 0, & \text{sicer} \end{cases},$$

kjer so $\{v_1, v_2, \ldots, v_n\} \in V$. V i-ti vrstici in j-tem stolpcu matrike sosednosti nastopa 1, če v G obstaja povezava $(i, j) \in E$, sicer pa 0. Matrika sosednosti je simetrična za neusmerjene grafe, pri enostavnih grafih pa so diagonalni elementi enaki 0.

Definicija 20. Utežen graf *(angl. weighted graph) je graf* $G = (V, E)$, *v katerem je vsaki povezavi* $e \in E$ *prirejena utež* $w(e) \in \mathbb{R}$. *Rečemo mu tudi omrežje (angl. network).*

Uteži lahko predstavljajo ceno, razdaljo ali katero drugo količino, odvisno od primera. Matrika sosednosti za utežen graf G v vrstici i in stolpcu j vsebuje $w\big((i, j)\big)$, če $(i, j) \in E$, sicer pa ∞.

Definicija 21. *Graf* $G' = (V', E')$ *je* podgraf *(angl. subgraph) grafa* $G = (V, E)$, *če velja:*

$$V' \subseteq V \ \wedge \ E' \subseteq E.$$

Podgraf G' *je* vpet, *če velja* $V' = V$.

Definicija 22. Sprehod S *v grafu* G, *dolžine* k, *je zaporedje vozlišč in povezav grafa, tako da je končno vozlišče vsake povezave enako začetnemu vozlišču naslednje povezave:*

$$S = (v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k),$$

kjer je $e_i = (v_{i-1}, v_i)$, *za* $\forall i = \{1, \ldots, k\}$. *Velikokrat zapišemo samo zaporedje povezav:* $S = (e_1, e_2, \ldots, e_k)$.

V uteženih grafih izračunamo dolžino sprehoda kot vsoto posameznih uteži povezav na tem sprehodu:

$$d(S) = \sum w(e_i), \ \forall i = \{1, \ldots, k\}.$$

Definicija 23. Pot *je sprehod, kjer so vsa vozlišča* $v_0, v_1, \ldots, v_k$ *med seboj različna.*

Če za sprehod velja, da so vsa vozlišča razen zadnjega med seboj različna in se končno vozlišče zadnje povezave v sprehodu ujema z začetnim vozliščem prve povezave $v_0 = v_k$, potem takemu sprehodu pravimo *cikel*. Graf brez ciklov je *acikličen*. Neusmerjen graf je *povezan*, če za vsak par vozlišč v grafu velja, da med njima obstaja pot. Usmerjenemu grafu s to lastnostjo pa pravimo *krepko povezan*.

Definicija 24. Razdaljo $d_G(u, v)$ *med vozliščema* $u, v \in V$ *v grafu* G *definiramo kot dolžino najkrajše poti od* u *do* v*. Če taka pot ne obstaja, za razdaljo vzamemo vrednost* ∞*.*

S tako definirano razdaljo postane povezan graf G metričen prostor (definicija (29)), za katerega veljajo lastnosti neenegativnosti, ločljivosti, simetričnosti in trikotniške neenakosti.

Matrika dolžin najkrajših poti je matrika, v kateri v i-ti vrstici in j-tem stolpcu nastopa dolžina najkrajše poti med vozliščema i in j.

Definicija 25. Topološko urejanje *usmerjenega acikličnega grafa* $G = (V, E)$ *je postopek linearnega urejanja oz. določanja vrstnega reda vozlišč grafa. Vozlišča* V *so topološko urejena takrat, kadar za poljubno povezavo* $(u, v) \in E$ *velja, da je* u *v tem urejanju pred* v*.*

Če torej graf narišemo tako, da vsa vozlišča nanizamo v vodoravni črti, so vozlišča topološko urejena, kadar so vse povezave usmerjene od leve proti desni.

2.4 Primerjava nizov

Ko merimo podobnost med objekti, naletimo na pojme, kot so: razdalja (angl. distance), podobnost (angl. similarity), različnost (angl. dissimilarity), primerjalka (angl. proxsimity) itd.

V tem razdelku bomo razjasnili te pojme, predstavili njihove definicije in lastnosti s področja teorije merjenja (razvrščanja v skupine; [Bre98], [Bat85])

ter s tem postavili temelje za nadaljnje raziskovanje področja mer podobnosti nizov.

2.4.1 Primerjalka, podobnost, različnost, razdalja

Primerjavo dveh nizov s in t količinsko ovrednotimo s *primerjalno funkcijo* ali krajše *primerjalko* p:

$$p : (s, t) \to \mathbb{R},\ \forall s, t \in \Sigma^*,$$

ki vsakemu paru nizov priredi neko realno število. Na posameznih področjih srečamo celo vrsto najrazličnejših primerjalk, zato je za njihovo predstavitev dobro določiti nekaj več lastnosti.

Definicija 26. *Preslikava* $p : \Sigma^* \times \Sigma^* \to \mathbb{R}$ *je* prema primerjalka *ali* mera različnosti *natanko takrat, ko zadošča pogoju:*

$$p(s, s) \leq p(s, t),\ \forall s, t \in \Sigma^*$$

in je obratna primerjalka *ali* mera podobnosti *natanko takrat, ko zadošča pogoju:*

$$p(s, s) \geq p(s, t),\ \forall s, t \in \Sigma^*.$$

Premo primerjalko označimo z d, obratno pa s s.

Definicija 27. *Mero podobnosti* s *imenujemo kratko* podobnost *natanko takrat, ko je simetrična:*

$$s(s, t) = s(t, s),\ \forall s, t \in \Sigma^*.$$

Definicija 28. *Mero različnosti* d *imenujemo kratko* različnost *natanko takrat, ko zadošča pogojem:*

$$\begin{aligned} d(s, t) \geq 0,\ \forall s, t \in \Sigma^* &\quad \textit{nenegativnost} \\ d(s, s) = 0,\ \forall s \in \Sigma^* &\quad \textit{ničelnost} \\ d(s, t) = d(t, s),\ \forall s, t \in \Sigma^* &\quad \textit{simetričnost.} \end{aligned}$$

Poleg tega je ugodno, če različnost zadošča še nekaterim izmed naslednjih lastnosti, kjer za $\forall s, t, u \in \Sigma^*$ velja:

$$d(s,t)=0 \implies d(s,u) = d(t,u) \quad \text{enakovrednost} \tag{2.10}$$

$$d(s,t)= 0 \implies s = t \quad \text{ločljivost} \tag{2.11}$$

$$d(s,t) \leq d(s,u) + d(u,t) \quad \text{trikotniška neenakost} \tag{2.12}$$

$$d(s,t) \leq \max(d(s,u), d(u,t)) \quad \text{ultrametrična neenakost} \tag{2.13}$$

Našteti pogoji so med seboj povezani:

$$(2.11) \implies (2.10) \impliedby (2.12) \impliedby (2.13)$$

Če različnost d iz definicije (28) zadošča še pogojema ločljivosti (2.11) in trikotniške neenakosti (2.12), ji pravimo *razdalja*.

Pojem *funkcije razdalje* ali *metrike* se pojavlja na mnogo različnih področjih in se pogosto uporablja za primerjavo dveh vektorjev ali n-teric. Zaradi njene pomembnosti lahko še enkrat formalno definiramo metriko oz. razdaljo d z vsemi pogoji na enem mestu:

Definicija 29. *Metričen prostor je neprazna množica Σ^* skupaj s preslikavo $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$, imenovano razdalja, z naslednjimi lastnostmi:*

$$d(s,t) \geq 0, \quad \forall s, t \in \Sigma^* \tag{2.14}$$

$$d(s,t) = 0 \implies s = t, \quad \forall s, t \in \Sigma^* \tag{2.15}$$

$$d(s,t) = d(t,s), \quad \forall s, t \in \Sigma^* \tag{2.16}$$

$$d(s,t) \leq d(s,u) + d(u,t), \quad \forall s, t, u \in \Sigma^* \tag{2.17}$$

Zadnja lastnost trikotniške neenakosti (2.17) je zelo uporabna in veliko algoritmov se zanaša na njeno veljavnost. Metrika, za katero ne velja trikotniška neenakost, zadošča pa ostalim pogojem, se imenuje *semi-metrika* oz. *semi-razdalja*.

2.4.2 Povezave med primerjalkami

Najpogosteje imajo različnosti za zalogo vrednosti ali interval $[0, 1]$ (normalizirana različnost) ali pa interval $[0, \infty]$. Zvezi med njima lahko dosežemo,

na primer, s preslikavama:

$$\frac{d}{(1-d)}\colon\ [0,1] \to [0,\infty] \quad \text{in} \quad \frac{d}{(1+d)}\colon\ [0,\infty] \to [0,1].$$

Obstajajo pa tudi primerjalke, ki ne zadoščajo gornjim zahtevam, na primer, mera, ki ima zalogo vrednosti $[-1,1]$. Njo prevedemo na interval $[0,1]$ s preslikavami:

$$\frac{(1+d)}{2},\ 1-|d|,\ \sqrt{(1-d^2)}.$$

Pri pretvarjanju ene primerjalke v drugo je zaželeno, da se "lepe" lastnosti ohranjajo, in da poleg tega obe primerjalki določata isto urejenost med nizi. Urejenost $\prec_d$, ki jo med pari nizov določa primerjalka d, lahko vpeljemo takole:

$$(s,t) \prec_d (u,w) \iff d(s,t) < d(u,w), \quad \forall s,t,u,w \in \Sigma^*.$$

Ohranjanje urejenosti nas privede do pojma enakovrednosti med primerjalkama.

Definicija 30. *Primerjalki d in ϕ sta* ekvivalentni*, kar zapišemo $d \sim \phi$, natanko takrat, ko je:*

$$\prec_d = \prec_\phi \quad \textit{ali} \quad \prec_d = \prec_\phi^{-1}.$$

Ekvivalenca primerjalk je ekvivalenčna relacija, kar pomeni, da zadovolji naslednje pogoje:

$$\begin{aligned} d \sim d &\ \text{ refleksivnost} \\ d \sim \phi \implies \phi \sim d &\ \text{ simetričnost} \\ d \sim \phi,\ \phi \sim \varphi \implies d \sim \varphi &\ \text{ tranzitivnost.} \end{aligned}$$

Izrek 1. *Naj bo d različnost in funkcija $f\colon \mathbb{R} \to \mathbb{R}$ na zalogi vrednosti različnosti d strogo monotono naraščajoča funkcija, za katero velja $f(0) = 0$, potem je različnost tudi funkcija $f \circ d$ in velja: $d \sim f \circ d$.*

Naj funkcija f poleg tega za poljubna argumenta x in y zadošča še pogoju subaditivnosti $f(x+y) \leq f(x) + f(y)$, potem, če je d razdalja, je razdalja tudi $f \circ d$.

Iz tega izreka (1) izhaja, da se lahko vsako mero podobnosti s predpisom

$$d(s,t) = -s(s,t),\ \forall s,t \in \Sigma^*$$

prevede v enakovredno mero različnosti. To pomeni, da vse, kar smo povedali o merah različnosti, velja tudi za mere podobnosti in obratno. In tudi v nadaljevanju naloge je dobro, če se zavedamo, da sta si pojma med sabo vsebinsko enakovredna, samo da se na njiju gleda z dveh različnih zornih kotov.

Poglavje 3

Pregled mer podobnosti oz. različnosti

V mnogih problemih s področja obdelave in analize nizov so prisotne potrebe po merjenju podobnosti oz. različnosti med dvema nizoma. Kot smo opisali v prejšnjem poglavju, lahko takšno mero definiramo s pomočjo funkcije razdalje ali pa s funkcijo podobnosti med nizoma.

Za ocenitev razdalje med dvema zaporedjema se lahko zaporedje dolžine n obravnava tudi kot vektor v $\mathbb{R}^n$. Pogoste metrike za vektorje iste dimenzije (zaporedja iste dolžine) so: Manhatnska razdalja, Evklidska razdalja ...

Pri problemih obdelave nizov se sestavni znaki pogosto ne interpretirajo kot numerične vrednosti in tudi primerjalni nizi so lahko različno dolgi. Zato se na tem področju za merjenje razdalj uporablja metrika, ki ovrednoti minimalne stroške za pretvorbo enega niza v drugega. Vsaki posamezni operaciji, vključeni v pretvorbo (zamenjava, vstavljanje, brisanje znaka), lahko pripišemo svojo ceno stroškov.

V tem poglavju bomo zelo na kratko predstavili nekaj skupin mer podobnosti, v nadaljevanju naloge pa se bomo omeljili in podrobneje pregledali skupino mer, ki temeljijo na operacijah urejanja.

3.1 Faktorizacija nizov

Nekaj razdalj med nizi lahko obravnavamo glede na faktorizacijo nizov. To so razdalja predpone, pripone in podniza. Njihov pomen je predvsem teoretične narave.

Definicija 31. Razdalja predpone: *za vsak* $u, v \in \Sigma^*$ *velja:*

$$d_{pref}(u, v) = |u| + |v| - 2 \times lcp(u, v),$$

kjer je $lcp(u, v)$ *dolžina najdaljše skupne predpone nizov* u *in* v *(angl. longest common prefix).*

Definicija 32. Razdalja pripone: *razdalja, definirana simetrično glede na razdaljo predpone, za vsak* $u, v \in \Sigma^*$ *velja:*

$$d_{suff}(u, v) = |u| + |v| - 2 \times lcsuff(u, v),$$

kjer $lcsuff(u, v)$ *predstavlja dolžino najdaljše skupne pripone nizov* u *in* v *(angl. longest common suffix).*

Definicija 33. Razdalja podniza: *rezdalja, definirana analogno prejšnjima dvema razdaljama, kjer za vsak* $u, v \in \Sigma^*$ *drži:*

$$d_{fact}(u, v) = |u| + |v| - 2 \times lcf(u, v),$$

kjer je $lcf(u, v)$ *dolžina najdaljšega skupnega podniza nizov* u *in* v *(angl. longest common factor).*

3.2 Mere na osnovi operacij urejanja

Ena od ključnih mer podobnosti oz. različnosti je *razdalja urejanja* (angl. edit distance). Razdalja urejanja $ED(s, t)$ je enaka najmanjšemu številu

operacij urejanja, ki transformirajo niz p v niz s. *Omejena razdalja urejanja* (angl. restricted edit distance) se izračuna kot minimalno število neprekrivajočih se operacij urejanja, ki ustvarijo dva niza enaka.

Če se med operacije urejanja štejejo vstavljanja, brisanja in zamenjave (substitucije) znakov ter imajo vse operacije strošek enak ena, potem takšno razdaljo imenujemo *Levenshteinova razdalja*. Če so poleg teh vključene tudi transpozicije, potem je to *Damerau-Levenshteinova razdalja*. *Hammingova razdalja* pa je različica, kjer so dovoljene le osnovne operacije zamenjave.

Če so dovoljena samo vstavljanja in brisanja, takšni razdalji rečemo *Indel razdalja*, ki je dualna problemu določitve *najdaljšega skupnega podzaporedja*.

Levenshteinovi razdalji, ki ima stroške operacij urejanja različne od ena, rečemo *posplošena Levenshteinova razdalja*.

Vse zgoraj naštete razdalje so primeri globalne poravnave nizov, obstaja pa tudi lokalna različica, ki temelji na meri podobnosti.

Primerjava mer podobnosti je predstavljena v Tabeli 3.1.

Mere podobnosti	Zamenjave	Brisanja	Vstavljanja	Transpozicije
Levenshteinova razdalja	1	1	1	/
posplošena Levenshteinova razdalja	$Sub(a, b)$	$Del(a)$	$Ins(b)$	/
Damerau-Levenshteinova razdalja	1	1	1	1
Hammingova razdalja	1	/	/	/
Indel razdalja	/	1	1	/

Tabela 3.1: Pregled mer podobnosti na osnovi operacij urejanja. V tabeli so označeni stroški posameznih pripadajočih operacij; vse mere (razen posplošena Levenshteinova razdalja) imajo stroške operacij vrednosti ena. Označba "/" pomeni, da posamezna mera ne dovoljuje določene operacije.

3.3 Q-grami

Naslednji pristop vključuje uporabo q-gramov. Stopnja podobnosti med nizi se lahko ocenjuje na podlagi primerjanja njihovih pripadajočih q-gramov. Večje kot je njihovo skupno število, bolj sta si niza podobna. Zmanjšana natančnost primerjave pri manjših vrednostih q mora biti uravnotežena s pretirano visokim številom možnih različnih q-gramov pri večjih vrednostih (abeceda Σ ima možnih $|\Sigma|^q$ različnih q-gramov). V praksi se največkrat uporabljajta vrednosti 2 in 3, imenovani tudi digram in trigram.

Ena izmed mer podobnosti na osnovi q-gramov je definirana kot razmerje med številom skupnih q-gramov med dvema nizoma in celotnim številom q-gramov v nizih. Druga funkcija razdalje je enaka vsoti absolutnih razlik med ustreznim številom pojavitev q-gramov v vsakem nizu. Imenujemo jo lahko tudi city-block razdalja med vektorji, ki predstavljajo nize. Komponente vsakega vektorja nam dajo število pojavitev v povezanem nizu od vseh $|\Sigma|^q$ možnih q-gramov. Za to funkcijo razdalje nam vrednost 0 ne zagotavlja enakosti nizov: imamo primer nizov *ada* in *dad*, ki sta različna, vendar sta njuna digrama popolnoma enaka, tako da je posledično njuna digram razdalja enaka 0.

3.4 Mere s skupnimi znaki

Nazadnje naj še omenimo mere, ki temeljijo na skupnih znakih nizov, ne glede na njihov vrstni red in mesto pojavitve. Obstaja mera podobnosti za skupne znake, kjer je večji poudarek na statistično redkejših znakih. Cene so dodeljene znakom v abecedi glede na njihovo relativno pogostost pojavitve in vrednost podobnosti se izračuna s seštevkom cen posameznih znakov, ki so skupni obema nizoma. Maksimalno vrednost te mere dobimo s primerjavo vzorca z njegovim anagramom ali s katerimkoli nizom, ki vsebuje vse znake iz abecede.

Poglavje 4

Mere na osnovi operacij urejanja

4.1 Razdalja urejanja

Zgodovina razdalje urejanja se začne z Dameraujem [Dam64], ki je predstavil statistiko pravopisnih napak in metodo za odpravo posameznih napačnih zapisov. Damerau se je osredotočil na vstavljanja, brisanja, substitucije in transpozicije enega znaka, ki so predstavljaje večino tipkarskih pravopisnih napak (okoli 80 procentov). Neodvisno je Levenshtein [Lev66] predlagal funkcijo različnosti, definirano kot minimalno število vstavljanj, brisanj in substitucij (vendar ne transpozicij) potrebnih za pretvorbo enega niza v drugega.

4.1.1 Sled urejanja in osnovne operacije urejanja

V bolj splošnem pogledu lahko en niz pretvorimo v drugega z zaporedjem atomarnih urejanj podnizov (glej Tabelo 4.1). To zaporedje operacij imenujemo *skripta ali sled urejanja* σ (angl. edit script, trace), atomarne trasformacije pa so *osnovne operacije urejanja* (angl. basic edit operations). Osnovno operacijo, ki zajema pretvorbo niza u v niz v, predstavimo kot $u \to v$. Množico osnovnih operacij urejanja označimo z $\mathbb{B}$.

Osnovne operacije urejanja enega samega znaka so običajno:

- **zamenjava** ($a \to b$)**:** znaka a iz niza u na danem položaju z znakom b iz niza v,
- **brisanje** ($a \to \varepsilon$)**:** znaka a iz niza u na danem položaju,
- **vstavljanje** ($\varepsilon \to b$)**:** znaka b iz niza v v niz u na dani položaj.

Včasih je $\mathbb{B}$ razširjen tudi z dodatno operacijo:

- **transpozicija** ($ab \to ba$)**:** ali obrnitev sosednjih znakov ab iz niza u na danem položaju, da dobimo znaka ba iz niza v.

Lastnost 1. *Predpostavimo, da* $\mathbb{B}$ *izpolnjuje naslednje lastnosti:*
- če $u \to v \in \mathbb{B}$*, potem tudi obratna operacija* $v \to u$ *pripada* $\mathbb{B}$ *(simetričnost);*
- $a \to a \in \mathbb{B}$ *(operacija identitete tudi pripada* $\mathbb{B}$*)*
- $\mathbb{B}$ *je poln:* $\forall$ *niz* s *in* t *obstaja sled operacij* σ*, ki pretvori* s *v* t*.*

$\mathbb{B}$ ni nujno končna množica.

4.1.2 Definicija razdalje urejanja

Podobnost med dvema nizoma lahko izrazimo z dolžino sledi operacij urejanja, ki naredi dva niza enaka.

Definicija 34 (Razdalja urejanja)**.** *Glede na dano množico osnovnih operacij urejanja* $\mathbb{B}$ *je* razdalja urejanja $ED(s,t)$ *enaka dolžini najkrajše sledi urejanja, ki transformira niz* s *v niz* t*. Najkrajša sled, ki transformira* s *v* t*, je* optimalna sled urejanja *(angl. optimal edit script).*

Definicija 35 (Levenshteinova razdalja)**.** Levenshteinova razdalja *je razdalja urejanja, kjer množica osnovnih operacij urejanja vsebuje samo vstavljanja, brisanja in substitucije (operacija identitete je poseben primer substitucije).*

Definicija 36 (Damerau-Levenshteinova razdalja)**.** Damerau-Levenshteinova razdalja *je razdalja urejanja, kjer so poleg osnovnih operacij vstavljanja, brisanja in substitucije zastopane še traspozicije.*

Definicija 37 (Hammingova razdalja)**.** Hammingova razdalja *je razdalja urejanja, kjer množica osnovnih operacij urejanja vsebuje samo substitucije. Poleg tega velja dodatna omejitev, da morata biti niza enake dolžine.*

Definicija 38 (Indel razdalja)**.** Indel razdalja *ali* razdalja podzaporedja *je razdalja urejanja, kjer množica osnovnih operacij urejanja vsebuje samo vstavljanja in brisanja.*

Razdaljo urejanja lahko interpretiramo tudi kot najmanjši strošek za transformacijo enega niza v drugi niz. Posplošitev lahko naredimo na dva načina.

Prvič, s stroškovno funkcijo $\delta : \Sigma \times \Sigma \to \mathbb{R}$ se lahko osnovnim operacijam urejanja $u \to v$ dodelijo posamezni stroški, kjer $\forall a, b \in \Sigma$ [WF74]:

- $\delta(a \to b)$ **ali** $Sub(a, b)$**:** strošek zamenjave znakov a z b,
- $\delta(a \to \varepsilon)$ **ali** $Del(a)$**:** strošek brisanja znaka a,
- $\delta(\varepsilon \to b)$ **ali** $Ins(b)$**:** strošek vstavljanja znaka b.

Na primeru Hammingove razdalje, kjer se upoštevajo samo operacije zamenjave, se določijo vrednosti $Del(a) = Ins(a) = +\infty$, pri Indel razdalji pa $Sub(a, b) = +\infty$, za vsak $a \in \Sigma$.

Stroškovno funkcijo $\delta()$ nato razširimo na sled urejanja $\sigma = a_1 \to b_1, a_2 \to b_2, \ldots, a_{|\sigma|} \to b_{|\sigma|}$ tako, da definiramo strošek sledi urejanja $cost(\sigma)$ (angl. cost function), ki predstavlja vsoto posameznih stroškov kriterijske funkcije:

Definicija 39.

$$cost(\sigma) = \sum_{i=1}^{|\sigma|} \delta(a_i \to b_i)$$

Razdalja od niza s do niza t je tako najmanjši strošek izmed vseh sledi urejanja, ki lahko pretvorijo niz s v niz t. Ta različica razdalje urejanja se ponavadi navaja kot *posplošena Levenshteinova razdalja* (generalized Levenshtein distance) ali tudi *utežena Levenshteinova razdalja* (weighted Levenshtein distance).

Drugič, množico osnovnih operacij urejanja $\mathbb{B}$ se lahko razširi tako, da so dovoljene tudi utežene zamenjave poljubnih nizov, ne pa samo operacije urejanja nad samo enim znakom [Ukk85]. Ta različica se imenuje *razširjena razdalja urejanja* (angl. extended edit distance). Na primer, $\mathbb{B}$ lahko vsebuje operacijo $x \to ks$. Potem je razširjena razdalja urejanja med nizoma "taxi"in "taksi"enaka 1, medtem ko je navadna Damerau-Levenshteinova razdalja enaka 2. Možno je tudi, da je stroškovna funkcija $\delta()$ odvisna od pozicij podniza.

Definicija 40. *Dani sta množica osnovnih operacij urejanja $\mathbb{B}$ in stroškovna funkcija $\delta()$, ki predpisuje strošek vsaki posamezni operaciji urejanja iz $\mathbb{B}$.* Splošna razdalja urejanja *med nizoma s in t je definirana kot minimalni strošek sledi urejanja σ, ki pretvori s v t:*

$$ED(s,t) = \min_{\sigma \in \mathcal{S}(s,t)} cost(\sigma)$$

kjer je $\mathcal{S}(s,t)$ množica vseh zaporedij osnovnih operacij, ki transformirajo s v t.

V Tabeli 4.1 je predstavljena razdalja urejanja na primeru nizov $s =$ TACTC in $t =$ GTCCGT.

Problem 1 (Optimalna sled urejanja nizov)**.**
Naloga: Končna abeceda Σ, niza s in t iz Σ^*, množica operacij urejanja $\mathbb{B}$ in stroškovna funkcija δ.
Dopustna rešitev: Sled urejanja σ, ki predstavlja zaporedje operacij urejanja iz $\mathbb{B}$, ki pretvorijo s v t.
Mera/ciljna funkcija:[1] *Strošek sledi urejanja $cost(\sigma)$, ki je vsota posameznih stroškov $\delta(u \to v)$, $u \to v \in \mathbb{B}$.*
Cilj: Minimizacija.

[1]Odvisno od množice $\mathbb{B}$ in stroškovne funkcije δ je mera lahko: Levenshteinova razdalja, Damerau-Levenshteinova razdalja, Hammingova razdalja, Indel razdalja, posplošena Levenshteinova razdalja ...

Operacija	Preoblikovan niz	Strošek
	TACTC	
vstavi G	**G**TACTC	1
zamenjaj T s T	**GT**ACTC	0
zamenjaj A z C	**GTC**CTC	1
zamenjaj C z C	**GTCC**TC	0
vstavi G	**GTCCG**TC	1
zamenjaj T s T	**GTCCGT**C	0
izbriši C	**GTCCGT**	1

Tabela 4.1: Prikaz razdalje urejanja. Zaporedje operacij urejanja za preoblikovanje niza TACTC v niz GTCCGT, kjer za vse znake $a, b \in \Sigma$ velja $Sub(a, a) = 0$, $Sub(a, b) = 1$, ko $a \neq b$, in $Del(a) = Ins(a) = 1$. Celotni strošek sledi urejanja $cost(\sigma)$ je $1+0+1+0+1+0+1 = 4$, torej je razdalja urejanja enaka ED(TACTC, GTCCGT)$= 4$.

Lastnost 2. *Predpostavimo, da stroškovna funkcija $\delta(u \to v)$ izpolnjuje naslednje lastnosti:*
- $\delta(u \to v) \in \mathbb{R}$ (stroškovna funkcija ima realno vrednost),
- $\delta(u \to v) = \delta(v \to u)$ (simetričnost),
- $\delta(u \to v) \geq 0$, $\delta(u \to u) = 0$, in $\delta(u \to v) = 0 \implies u = v$ (pozitivna definitnost),
- $\forall \gamma > 0$ je množica osnovnih operacij $\{u \to v \in \mathbb{B} \mid \delta(u \to v) < \gamma\}$ končna (končnost podmnožice osnovnih operacij urejanja, katerih stroški so navzgor omejeni).

Zadnja lastnost drži avtomatsko za vsako končno $\mathbb{B}$.

Trditev 2. *Iz lastnosti* (1) *in* (2) *sledi:*
- Za vsak par nizov s in t obstaja sled urejanja σ z minimalnimi stroški.
- Splošna razdalja urejanja $ED(s, t)$ iz definicije (40) *je metrika [WF74].*

Dokaz. Za dokaz, da je $ED(s, t)$ metrika oz. razdalja, moramo pokazati, da $ED(s, t)$ obstaja in je pozitivno definitna, simetrična in velja za njo

trikotniška neenakost. [Boy11]

Iz lastnosti (2) sledi, da je stroškovna funkcija nenegativna in da ima strošek nič samo operacija identitete. Zato se lahko brez škode za splošnost osredotočimo na sledi urejanja, ki ne vsebujejo operacij identitete. Torej, če je $s = t$, je edina optimalna sled urejanja (ki ne vsebuje operacij identitete) prazna in ima strošek nič. Če je $s \neq t$, iz polnosti množice osnovnih operacij urejanja $\mathbb{B}$ sledi, da obstajajo ena ali več sledi urejanja, ki pretvorijo s v t. Vse takšne sledi urejanja vsebujejo operacije urejanja s strogo pozitivnimi stroški.

Naj bo γ strošek poljubne sledi, ki pretvori s v t. Upoštevajmo množico sledi urejanja $\mathcal{S}$, ki pretvorijo s v t, in katere stroški so navzgor omejeni z γ. $\mathcal{S}$ je neprazna in vsebuje operacije urejanja s pozitivnimi stroški, manjšimi od γ. Množica osnovnih operacij urejanja $\mathbb{B}$, katerih stroški so navzgor omejeni z γ, je končna, kar dokazuje, da je tudi $\mathcal{S}$ končna. Ker je $\mathcal{S}$ neprazna in končna, obstaja sled urejanja z minimalnimi (pozitivnimi) stroški in je vsebovana v $\mathcal{S}$. Torej, $ED(s,t) > 0$ za $s \neq t$, kar pomeni, da je razdalja urejanja pozitivno definitna.

Simetričnost razdalje urejanja sledi iz simetričnosti stroškovne funkcije in simetričnosti osnovnih operacij urejanja iz $\mathbb{B}$. Za dokaz simetričnosti razdalje urejanja upoštevajmo optimalno sled σ, ki pretvori s v t, in ustrezno obrnjeno sled σ_r, ki transformira t v s. Če v sledi σ obrnemo vrstni red operacij ter zamenjamo operacije vstavljanja in brisanja, dobimo sled σ_r. Sledi $cost(\sigma) = cost(\sigma_r)$.

Dokažimo še trikotniško neenakost. Naj bo σ_1 optimalna sled, ki pretvori s v t, σ_2 optimalna sled, ki pretvori t v w in sestavljena sled $\sigma_1\sigma_2$, ki pretvori s v w. Iz $cost(\sigma_1\sigma_2) = cost(\sigma_1) + cost(\sigma_2) = ED(s,t) + ED(t,w)$ in $cost(\sigma_1\sigma_2) \geq ED(s,w)$ sledi $ED(s,t) + ED(t,w) \geq ED(s,w)$. □

Razdalja urejanja je metrika, tudi če za stroškovno funkcijo $\delta()$ ne velja trikotniška neenakost. Ker lahko ima zaporedje prekrivajočih operacij za pretvorbo niza u v niz v nižji strošek kot direktna operacija $\delta(u \to v)$, je lahko $\delta(u \to v)$ večja od $ED(u,v)$. Primer: imamo abecedo $\{a,b,c\}$ in stroškovno

funkcijo $\delta()$, za katero velja simetričnost, ne pa trikotniška neenakost:

$$\begin{aligned}\delta(a \to c) &= \delta(b \to c) = 1,\\ \delta(a \to \varepsilon) = \delta(b \to \varepsilon) &= \delta(c \to \varepsilon) = 2,\\ \delta(a \to b) &= 3.\end{aligned}$$

Vidimo, da $3 = \delta(a \to b) > \delta(a \to c) + \delta(c \to b) = 2$. Optimalna sled urejanja $(a \to c,\ c \to b)$ torej transformira a v b za ceno 2.

Zato razjasnimo zadostne pogoje stroškovne funkcije, ki so potrebni za to, da je razdalja urejanja metrika:

Trditev 3. *Razdalja urejanja ED iz definicije* (40) *je razdalja nad* Σ^* *natanko takrat, ko je* $Sub(a, b)$ *razdalja nad* Σ *in* $Del(a) = Ins(a) > 0$ *za vsak* $a \in \Sigma$.

Dokaz je zelo podoben tistemu za trditev (2), najdemo ga lahko v [CHL01].

4.1.3 Omejena razdalja urejanja

Trikotniška neenakost razdalje urejanja omogoča, da je takšna razdalja v praksi veliko bolj uporabna, saj se jo lahko uporablja pri vseh metodah, ki temeljijo na metričnih prostorih. Vendar je problem minimizacije nad množico vseh možnih prekrivajočih se operacij urejanja lahko težek. Za rešitev računske kompleksnosti problema se zato uporablja raje mera podobnosti oz. različnosti, ki je definirana kot minimalen strošek *omejene sledi urejanja* (angl. restricted edit script). Omejena sled urejanja ne vsebuje prekrivajočih se operacij urejanja in ne spreminja večkrat istega podniza. Takšni razdalji rečemo *omejena razdalja urejanja* (angl. restricted edit distance).

Problem 2 (Optimalna omejena sled urejanja nizov)**.**
Naloga: Končna abeceda Σ*, niza* s *in* t *iz* Σ^**, množica operacij urejanja* $\mathbb{B}$ *in stroškovna funkcija* δ*.*
Dopustna rešitev: Omejena sled urejanja σ*, ki predstavlja zaporedje neprekrivajočih se operacij urejanja iz* $\mathbb{B}$*, ki pretvorijo* s *v* t*.*

Mera/ciljna funkcija:[2] *Strošek omejene sledi urejanja* $cost(\sigma)$*, ki je vsota posameznih stroškov* $\delta(u \to v)$*,* $u \to v \in \mathbb{B}$*.*
Cilj: Minimizacija.

Trditev 4. *Vsaka neomejena razdalja urejanja je spodnja meja za ustrezno omejeno razdaljo urejanja.*

Trditev 5. *Omejena Levenshteinova razdalja je enaka neomejeni Levenshteinovi razdalji.*

Dokaz sledi iz opazke, da optimalna sled urejanja vsebuje enoznakovna brisanja, vstavljanja ali zamenjave, ki nikoli ne spremenijo istega znaka dvakrat.

Trditev 6. *Neomejena Damerau-Levenshteinova razdalja ni enaka omejeni Damerau-Levenshteinovi razdalji, saj velja, da omejena Damerau-Levenshteinova razdalja ni metrika, ampak je samo semi-metrika, ker za njo ne velja trikotniška neenakost.*

Dokaz. Damerau-Levenshteinova razdalja obravnava transpozicije (dvoznakovni preobrat dveh sosednjih znakov) kot osnovne operacije urejanja. Za dokaz uporabimo primer, kjer prepoved spreminjanja že prenesenih znakov prikaže razliko med omejeno in neomejeno Damerau-Levenshteinovo razdaljo.

Vzemimo nize *ab*, *ba* in *acb*. Najkrajša neomejena sled urejanja, ki pretvori *ba* v *acb* vsebuje dve operaciji ($ba \to ab,\ \varepsilon \to c$): najprej zamenjavo *a* in *b*, potem pa vstavljanje *c*. Pri tem vstavljanje spremeni že spremenjen niz. Če pa naknadne spremembe istega niza niso dovoljene, najkrajša sled urejanja pretvori *ba* v *acb* s tremi operacijami ($b \to \varepsilon,\ \varepsilon \to c,\ \varepsilon \to b$). Tako ima neomejena razdalja urejanja vrednost dva, omejena razdalja pa je enaka tri.

[2]Odvisno od množice $\mathbb{B}$ in stroškovne funkcije δ je tudi tu mera lahko: Levenshteinova razdalja, Damerau-Levenshteinova razdalja, Hammingova razdalja, Indel razdalja, posplošena Levenshteinova razdalja ...

Damerau-Levenshteinova razdalja od ab do ba, kot tudi od ab do acb ima vrednost ena. Tako da omejena Damerau-Levenshteinova razdalja ne izpolnjuje trikotniške neenakosti zaradi:

$$2 = ED(ab, ba) + ED(ab, acb) < ED(ba, acb) = 3.$$

□

Problem učinkovitega izračuna neomejene Damerau-Levenshteinove razdalje sta rešila Lowrance in Wagner [WL75].

4.2 Vizualizacije primerjav

V nadaljevanju bodo predstavljeni postopki vizualizacije, ki se pogosto uporabljajo za primerjanje nizov. Med njimi so: poravnave nizov, grafi urejanja ter točkovni diagrami [CZ08], [JP04].

4.2.1 Optimalna poravnava

Poravnava dveh nizov $s, t \in \Sigma^*$ je vizualen način predstavitve njune podobnosti (glej Tabelo 4.2).

Naj bosta s in t dva niza, sestavljena iz znakov abecede Σ. Poravnava nizov s in t je par nizov $(\tilde{s}, \tilde{t})$, ki jih dobimo z vstavljanjem presledkov[3] "–" v s in t. Poravnava $\alpha = (\tilde{s}, \tilde{t})$ mora zadovoljiti lastnostim:

- $|\tilde{s}| = |\tilde{t}|$
- odstranitev vseh presledkov iz $\tilde{s}$ nam da s
- odstranitev vseh presledkov iz $\tilde{t}$ nam da t
- za vsak i, vsaj eden izmed $\tilde{s}[i]$ in $\tilde{t}[i]$ ni presledek

Bolj formalno lahko zapišemo:

[3] Včasih se uporablja tudi izraz *luknja* (angl. hole).

Definicija 41. *Globalna poravnava nizov s in t je enaka paru nizov $\alpha = (\tilde{s}, \tilde{t})$ nad abecedo iz parov znakov, bolj natančno: $(\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \setminus \{(-,-)\}$. Torej, če je $\alpha = (\tilde{s}, \tilde{t})$ globalna poravnava, dolžine n, nizov s in t, imamo:*

$$\begin{aligned}(\tilde{s}, \tilde{t}) &= (\tilde{s}[1], \tilde{t}[1])(\tilde{s}[2], \tilde{t}[2]) \dots (\tilde{s}[n], \tilde{t}[n]),\\ \tilde{s} &= \tilde{s}[1]\tilde{s}[2] \dots \tilde{s}[n],\\ \tilde{t} &= \tilde{t}[1]\tilde{t}[2] \dots \tilde{t}[n],\end{aligned}$$

kjer je $\tilde{s}[i], \tilde{t}[i] \in \Sigma \cup \{-\},\ \forall i = 1, 2 \dots, n$. Poravnava:

$$\alpha = (\tilde{s}, \tilde{t}) = (\tilde{s}[1], \tilde{t}[1])(\tilde{s}[2], \tilde{t}[2]) \dots (\tilde{s}[n], \tilde{t}[n]),$$

dolžine n, je lahko označena tudi z:

$$\binom{\tilde{s}[1]}{\tilde{t}[1]}\binom{\tilde{s}[2]}{\tilde{t}[2]} \dots \binom{\tilde{s}[n]}{\tilde{t}[n]},$$

ali

$$\begin{pmatrix}\tilde{s}[1] & \tilde{s}[2] & \dots & \tilde{s}[n]\\ \tilde{t}[1] & \tilde{t}[2] & \dots & \tilde{t}[n]\end{pmatrix}.$$

Izraz "globalna" poravnava poudari dejstvo, da sta v poravnavo vključena celotna niza, za razliko od lokalne poravnave, ki jo bomo obravnavali kasneje.

Poravnava ustvari povezavo med znakoma $\tilde{s}[i]$ in $\tilde{t}[i]$, ki zasedata isto mesto i; rečemo, da sta znaka $\tilde{s}[i]$ in $\tilde{t}[i]$ poravnana v skladu z $\alpha = (\tilde{s}, \tilde{t})$. To pomeni, da velja: $\tilde{s}[i] \to \tilde{t}[i] \in \mathbb{B}$.

Poravnan par (a, b) (angl. aligned pair), kjer $a, b \in \Sigma$, pomeni zamenjavo znaka b z znakom a. Poravnan par tipa $(a, -)$, $a \in \Sigma$, predstavlja brisanje znaka a. Na zadnje, poravnan par $(-, b)$, $b \in \Sigma$, pomeni vstavljanje znaka b.

Strošek poravnanega para definiramo za vsak $a, b \in \Sigma$:

$$\begin{aligned}cost(a, b) &= Sub(a, b),\\ cost(a, -) &= Del(a),\\ cost(-, b) &= Ins(b).\end{aligned}$$

Operacija	Poravnan par	Strošek
vstavi G	(−, G)	1
zamenjaj T s T	(T, T)	0
zamenjaj A z C	(A, C)	1
zamenjaj C z C	(C, C)	0
vstavi G	(−, G)	1
zamenjaj T s T	(T, T)	0
izbriši C	(C, −)	1

Tabela 4.2: Nadaljevanje primera iz Tabele 4.1. Vsaki operaciji urejanja je dodeljen ustrezen poravnan par (a, b), kjer $a, b \in \Sigma$. Temu pripada sledeča poravnava:

$$\begin{pmatrix} - & \mathrm{T} & \mathrm{A} & \mathrm{C} & - & \mathrm{T} & \mathrm{C} \\ \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{C} & \mathrm{G} & \mathrm{T} & - \end{pmatrix}.$$

Primer poravnave nizov $s = \mathrm{TACTC}$ in $t = \mathrm{GTCCGT}$ lahko vidimo v Tabeli 4.2.

Poravnava nizov s in t je torej ekvivalentna omejeni sledi urejanja:

$$\sigma = \tilde{s}[1] \to \tilde{t}[1],\ \tilde{s}[2] \to \tilde{t}[2], \ldots, \tilde{s}[n] \to \tilde{t}[n].$$

Definirajmo *strošek poravnave* (angl. cost of alignment) kot strošek ustrezne sledi urejanja in jo označimo $cost(\alpha)$:

$$cost(\alpha) = \sum_{i=1}^{n} \delta\big(\tilde{s}[i] \to \tilde{t}[i]\big) \tag{4.1}$$

Definicija 42. *Optimalna poravnava je poravnava z minimalnim stroškom:*

$$align(s, t) = \min_{\alpha \in \mathcal{A}(s,t)} cost(\alpha), \tag{4.2}$$

kjer je $\mathcal{A}(s, t)$ množica vseh poravnav nizov s in t.

Problem 3 (Globalna optimalna poravnava nizov)**.**
Naloga: Končna abeceda Σ, niza s in t iz Σ^* in stroškovna funkcija $cost(a, b)$, $a, b \in \Sigma$.
Dopustna rešitev: Globalna poravnava $\alpha = (\tilde{s}, \tilde{t})$ nizov s in t; tj. $|\tilde{s}| = |\tilde{t}|$, odstranitev presledkov iz $\tilde{s}$ nam da s in iz $\tilde{t}$ nam da t ter $\forall i$ vsaj eden izmed $\tilde{s}[i]$ in $\tilde{t}[i]$ ni presledek.
Mera/ciljna funkcija: Strošek globalne poravnave $cost(\alpha)$ nizov s, t, ki je enak vsoti stroškov posameznih poravnanih parov $cost(a, b)$.
Cilj: Minimizacija.

Število poravnav dveh nizov je eksponentno. Naslednji predlog določa to količino za poseben tip poravnav in tako daje spodnjo mejo za skupno število poravnav.

Trditev 7. *Naj bosta niza $s, t \in \Sigma^*$, posameznih dolžin m in n, kjer je $m \leq n$. Število poravnav nizov s in t, ki ne vsebujejo zaporednih brisanj znakov iz s, je $\binom{2n+1}{m}$.*

Dokaz. Lahko vidimo, da je vsaka poravnava enolično določena z zamenjavami na n pozicijah niza t in z $n + 1$ pozicijami brisanj med znaki niza t (štejemo eno mogoče brisanje pred $t[0]$ in eno po $t[n - 1]$).

Poravnava je tako določena z izbiro m zamenjav ali brisanj na $2n + 1$ mogočih mestih, kar nam da napovedan rezultat. □

Lahko se vidi, da obstaja preslikava med množico omejenih sledi urejanja in množico poravnav: vsaka omejena sled urejanja z minimalnim stroškom predstavlja poravnavo z minimalnim stroškom in obratno.

Trditev 8. *Problem iskanja* Optimalne omejene sledi urejanja nizov *s in t je enakovreden problemu iskanja* globalne optimalne poravnave nizov *s, t:*

$$reED(s, t) = align(s, t)$$

Tako lahko nadomestimo nalogo iskanja Optimalne omejene sledi urejanja nizov z nalogo iskanja stroška globalne optimalne poravnave nizov.

4.2.2 Graf urejanja

Poravnava se lahko prevede v obliko grafa. V ta namen vpeljemo pojem *grafa urejanja* $G(s,t)$ (angl. edit graph) dveh nizov $s,t \in \Sigma^*$, ustreznih dolžin m in n kot sledi (glej Sliko 4.1).

Z V označimo množico vozlišč grafa $G(s,t)$ in z E njegovo množico usmerjenih povezav. Povezave so označene s funkcijo "label", katere vrednosti so poravnani pari, in so ovrednotene s stroški teh parov.

Množica vozlišč V je:

$$V = \{-1, 0, \ldots, m-1\} \times \{-1, 0, \ldots, n-1\},$$

množica povezav E je:

$$\begin{aligned} E = &\Big\{\Big((i-1, j-1), (i,j)\Big) \colon\ (i,j) \in V \text{ in } i \neq -1 \text{ in } j \neq -1\Big\} \\ &\cup \Big\{\Big((i-1, j), (i,j)\Big) \colon\ (i,j) \in V \text{ in } i \neq -1\Big\} \\ &\cup \Big\{\Big((i, j-1), (i,j)\Big) \colon\ (i,j) \in V \text{ in } j \neq -1\Big\}, \end{aligned}$$

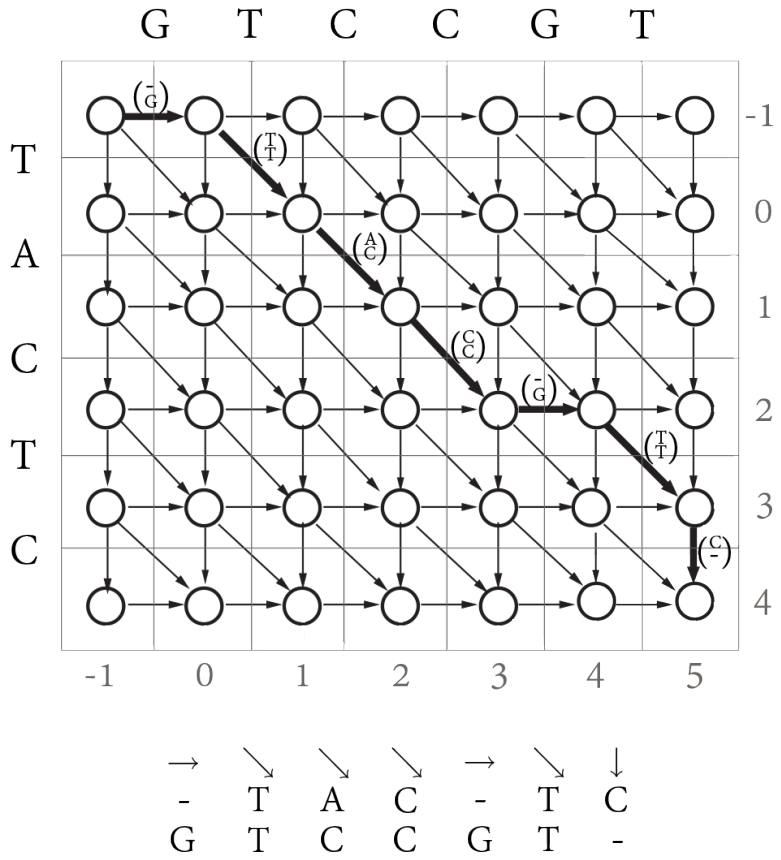
in funkcija:

$$label \colon\ E \to \big(\Sigma \cup \{-\}\big) \times \big(\Sigma \cup \{-\}\big) \setminus \{(-,-)\}$$

je definirana z:

$$\begin{aligned} label\Big((i-1, j-1), (i,j)\Big) &= \big(s[i], t[j]\big), \\ label\Big((i-1, j), (i,j)\Big) &= \big(s[i], -\big), \\ label\Big((i, j-1), (i,j)\Big) &= \big(-, t[j]\big). \end{aligned}$$

Vsaka pot, ki gre iz izhodišča $(-1,-1)$ do končnega vozlišča $(m-1, n-1)$, predstavlja poravnavo nizov s in t. To lahko vidimo na primeru grafa urejanja na Sliki 4.1. Torej če izberemo za začetno stanje $(-1,-1)$ in za končno stanje $(m-1, n-1)$, lahko graf urejanja $G(s,t)$ postane avtomat, ki je zmožen prepoznati vse poravnave nizov s in t. Strošek povezave $e \in E$ grafa $G(s,t)$ je enak vrednosti posamezne oznake, torej $cost(label(e))$.



Slika 4.1: Graf urejanja G(TACTC, GTCCGT) brez prikazanih stroškov, ki predstavlja nadaljevanje primera iz Tabele 4.1 in Tabele 4.2. Vsaka pot, ki poteka iz vozlišča $(-1,-1)$ do vozlišča $(m-1, n-1)$, predstavlja poravnavo med nizoma TACTC in GTCCGT. Označena pot ustreza optimalni poravnavi.

Problem 4 (Najcenejša pot v grafu urejanja)**.**
Naloga: Končna abeceda Σ, graf urejanja $G(s,t)$ nizov $s,t \in \Sigma^*$, sestavljen iz množice vozlišč V, množice povezav E in funkcije "label" ter stroškovna funkcija $cost(a,b)$, $a,b \in \Sigma$.
Dopustna rešitev: Pot v grafu $G(s,t)$, ki gre od izhodišča $(-1,-1)$ do končnega vozlišča $(m-1,n-1)$.
Mera/ciljna funkcija: Strošek poti v grafu $G(s,t)$, ki je enak vsoti stroškov posameznih povezav $cost(label(e))$, $e \in E$.
Cilj: Minimizacija.

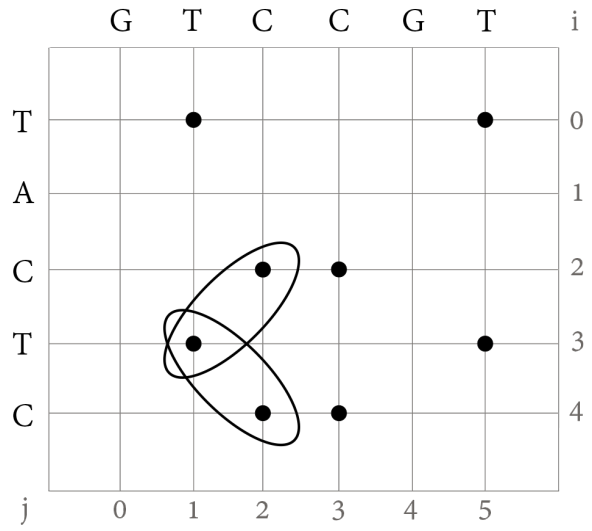
Trditev 9. *Izračun optimalne poravnave $align(s,t)$ ali izračun omejene razdalje urejanja $reED(s,t)$ je enak določitvi poti minimalnega stroška, ki se začenja v $(-1,-1)$ in konča v $(m-1,n-1)$ v grafu $G(s,t)$.*

$$align(s,t) = reED(s,t) = d_G\big((-1,-1),(m-1,n-1)\big)$$

Poti minimalnega stroška so vsaka posamezno v relaciji z optimalnimi poravnavami nizov s in t. Ker je graf $G(s,t)$ acikličen, je možno najti pot minimalnega stroška z upoštevanjem vsakega vozlišča samo enkrat. Zaradi tega zadošča pregledati vozlišča v G glede na topološko urejenost. Takšno urejenost lahko dosežemo s pregledom vozlišč, stolpec po stolpec, iz leve proti desni in od zgoraj navzdol, znotraj vsakega stolpca. Rezultat lahko dobimo tudi s pregledom vozlišč, vrstico po vrstico, od zgoraj navzdol in od leve proti desni, znotraj vsake vrstice, ali na primer s pregledovanjem glede na antidiagonale. Problem lahko rešimo z dinamičnim programiranjem, ki bo predstavljeno kasneje.

4.2.3 Točkovni diagram

Obstaja zelo enostavna metoda za označitev podobnosti med dvema nizoma s in t, pripadajočih dolžin m in n. V ta namen opredelimo tabelo *Dot*, velikosti $m \times n$, imenovano *točkovni diagram* (angl. dotplot) nizov s in t. Vrednosti tabele *Dot* so definirane za vsak položaj i v nizu s in vsak položaj



Slika 4.2: Točkovni diagram med nizoma s = TACTC in t = GTCCGT. Črne pike so postavljene na mestih (i, j), kjer je $s[i] = t[j]$. Na diagramu sta označeni diagonala in antidiagonala, ki predstavljata podobnost podnizov (diagonala $\big((3,1),(4,2)\big)$ nam pove, da je pripona TC $\sqsupset$ s podniz od t; antidiagonala $\big((3,1),(2,2)\big)$ pa, da se faktor CT $\preceq_{fakt}$ s pojavi v obratnem vrstnem redu v t).

j v nizu t:

$$Dot[i,j] = \begin{cases} \textbf{true} & \text{če } s[i] = t[j], \\ \textbf{false} & \text{sicer.} \end{cases}$$

Točkovni diagram predstavimo tako, da na mrežo postavimo točke, ki označujejo vrednosti **true**. Območja podobnosti med dvema nizoma so prikazana kot zaporedja točk na diagonalah mreže. Primer je viden na Sliki 4.2.

S točkovnega diagrama je mogoče s povezovanjem zaporedij točk izpeljati globalno poravnavo dveh nizov. Diagonalne povezave ustrezajo zamenjavam, vodoravne povezave ustrezajo vstavljanjem in navpične povezave brisanjem. Globalne poravnave tako presdtavljajo poti v mreži, ki se začnejo blizu zgornjega levega kota in končajo blizu spodnjega desnega kota.

4.3 Izračun razdalje urejanja

V nadaljevanju bomo podali formalno definicijo in pregled klasičnega algoritma na osnovi dinamičnega programiranja za izračun Levenshteinove razdalje in njegove razširitve (Lowrance in Wagner [WL75]).

Algoritem na osnovi dinamičnega programiranja za izračun stroška optimalne poravnave je neodvisno odkrilo več raziskovalcev z različnih področij, kot je prepoznavanje govora in bioinformatika. Kljub zgodnjim odkritjem je bil algoritem splošno nepoznan pred objavo Wagnerja in Fischerja [WF74].

Osnovni princip algoritma je izračunati strošek poravnave nizov s in t z uporabo stroškov poravnav njunih predpon. Imamo predpono $s[1..i]$, dolžine i, in predpono $t[1..j]$, dolžine j. Predpostavimo, da je $\alpha = (\tilde{s}, \tilde{t})$ optimalna poravnava nizov $s[1..i]$ in $t[1..j]$, katere strošek je $C_{i,j}$.

Iz enačbe (4.1) in definicije o poravnavi (41) sledi, da se lahko $C_{i,j}$ izračuna s pomočjo sledeče rekurzivne formule (Ukkonen [Ukk85], Veronis [Vér88]):

$$\begin{aligned} C_{0,0} &= 0, \\ C_{i,j} &= \min\left\{\delta\big(\tilde{s}[i'..i] \to \tilde{t}[j'..j]\big) + C_{i'-1,j'-1} \;\middle|\; \tilde{s}[i'..i] \to \tilde{t}[j'..j] \in \mathbb{B}\right\}. \end{aligned} \tag{4.3}$$

Zgornja rekurzija je primer rešitve s pomočjo dinamičnega programiranja. Množica $(|s|+1)\cdot(|t|+1)$ števil $C_{i,j}$ se pogosto imenuje *matrika dinamičnega programiranja* (angl. dynamic programming matrix).

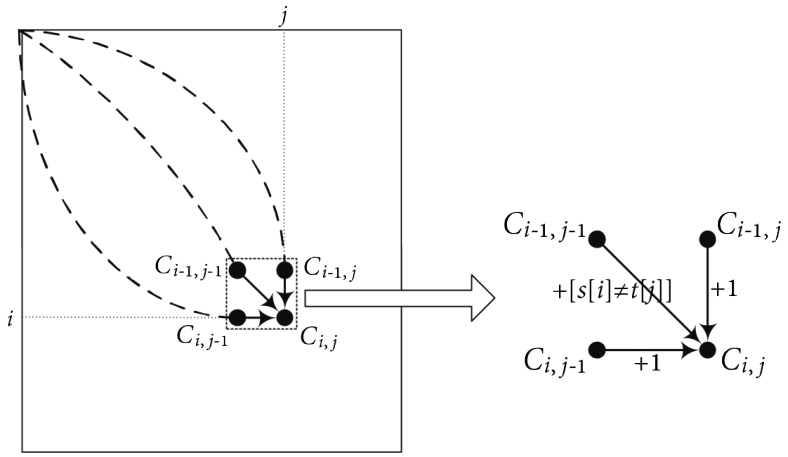
Poglejmo sedaj primer Levenshteinove razdalje, kjer $\tilde{s}[i'..i] \to \tilde{t}[j'..j]$ predstavlja strošek vstavljanja, brisanja ali zamenjave enega znaka. Zato:

$$\delta\big(\tilde{s}[i'..i] \to \tilde{t}[j'..j]\big) = \Big[\tilde{s}[i'..i] \neq \tilde{t}[j'..j]\Big],$$

če je pogoj izpolnjen, je rezultat 1, sicer pa 0.

Obstajajo tri možne kombinacije i' in j', ki ustrezajo brisanju, vstavljanju in zamenjavi:

$$\begin{aligned} i' = i-1 \;\text{ in }\; j' &= j \\ i' = i \;\text{ in }\; j' &= j-1 \\ i' = i-1 \;\text{ in }\; j' &= j-1. \end{aligned}$$



Slika 4.3: Slika pojasni rekurzijo tako, da prikaže tri možne povezave za vstop v vozlišče (i, j), ko $i, j > 0$. Vrednost minimalne poti, ki vstopi navpično v vozlišče (i, j) iz vozlišča $(i-1, j)$, je enaka vsoti vrednosti minimalne poti, ki vstopi v vozlišče $(i-1, j)$, in vrednosti povezave $(i-1, j) \to (i, j)$. Torej je vrednost minimalne poti, ki vstopi v (i, j) in se konča z operacijo brisanja, enaka $C_{i-1,j} + 1$. Podobno za vrednost minimalne poti, ki vstopi v (i, j) iz $(i, j-1)$ horizontalno, velja $C_{i,j-1}+1$ in za vrednost minimalne poti, ki vstopi v (i, j) iz $(i-1, j-1)$ diagonalno, sledi $C_{i-1,j-1} + \big[s[i] \neq t[j]\big]$. Za izračun $C_{i,j}$ torej izberemo minimalno vrednost izmed teh treh možnosti. Vir slike: [CZ08].

Glede na zgornje primere lahko zapišemo rekurzijo (4.3) za Levenshteinovo razdaljo takole (glej Sliko 4.3):

$$C_{i,j} = \min \begin{cases} 0, & \text{če } i = j = 0 \\ C_{i-1,j} + 1, & \text{če } i > 0 \\ C_{i,j-1} + 1, & \text{če } j > 0 \\ C_{i-1,j-1} + \big[s[i] \neq t[j]\big], & \text{če } i, j > 0 \end{cases} \tag{4.4}$$

Glede na trditev (5) je neomejena Levenshteinova razdalja enaka omejeni Levenshteinovi razdalji. Po drugi strani pa je omejena razdalja urejanja

enaka strošku optimalne poravnave. Torej rekurzija (4.4) izračuna neomejeno Levenshteinovo razdaljo. Spodnja enostavna posplošitev rekurzije (4.4) predstavlja izračun omejene Damerau-Levenshteinove razdalje:

$$C_{i,j} = \min \begin{cases} 0, & \text{če } i = j = 0 \\ C_{i-1,j} + 1, & \text{če } i > 0 \\ C_{i,j-1} + 1, & \text{če } j > 0 \\ C_{i-1,j-1} + \big[s[i] \neq t[j]\big], & \text{če } i, j > 0 \\ C_{i-2,j-2} + 1, & \text{če } s[i]{=}t[j-1],\ s[i-1]{=}t[j],\ i, j{>}1 \end{cases} \tag{4.5}$$

Glede na trditev (6) omejena Damerau-Levenshteinova razdalja ni vedno enaka neomejeni Damerau-Levenshteinovi razdalji. Na primer razdalja, izračunana z rekurzijo (4.5) med nizoma "*ba*" in "*acb*", je enaka 3, medtem ko je neomejena Damerau-Levenshteinova razdalja enaka 2.

Kot sta pokazala Lowrance in Wagner [WL75], se neomejeno Damerau-Levenshteinovo razdaljo da izračunati kot strošek omejene razdalje urejanja (tudi strošek optimalne poravnave), kjer $\mathbb{B}$ poleg osnovnih operacij urejanja (vstavljanja, brisanja, zamenjave) vsebuje tudi operacijo $aub \to bva$, ki stane $|u| + |v| + 1$. Glede na to definicijo in splošno rekurzijo (4.3) lahko neomejeno Damerau-Levenshteinovo razdaljo zapišemo:

$$C_{i,j} = \min \begin{cases} 0, & \text{če } i = j = 0 \\ C_{i-1,j} + 1, & \text{če } i > 0 \\ C_{i,j-1} + 1, & \text{če } j > 0 \\ C_{i-1,j-1} + \big[s[i] \neq t[j]\big], & \text{če } i, j > 0 \\ \min\limits_{\substack{0<i'<i,\ 0<j'<j, \\ s[i]=t[j'],\ s[i']=t[j]}} C_{i'-1,j'-1} + (i - i') + (j - j') - 1 \end{cases} \tag{4.6}$$

V nadaljevanju bomo natančneje opisali postopek izračuna posplošene Levenshteinove razdalje, saj če znamo izvesti splošnejšo različico algoritma,

je potem prevedba na zgornji primer Levenshteinove razdalje (4.4) zelo trivialna.

4.3.1 Izračun posplošene Levenshteinove razdalje

Predstavimo sedaj postopek za izračun posplošene oz. utežene Levenshteinove razdalje med nizoma s in t z metodo dinamičnega programiranja. Metoda temelji na pomnjenju vmesnih rezultatov, da bi se s tem izognili njihovim ponovnim izračunom.

Za niza $s, t \in \Sigma^*$, sledečih dolžin m in n, definiramo tabelo T z $m+1$ vrsticami in $n+1$ stolpci:

$$T[i,j] = ED(s[0..i], t[0..j])$$

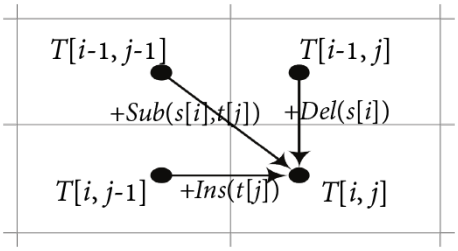
za $i = -1, 0, \ldots, m-1$ in $j = -1, 0, \ldots, n-1$. Torej je $T[i,j]$ tudi minimalen strošek poti od $(-1,-1)$ do (i,j) v grafu urejanja $G(s,t)$.

Za izračun $T[i,j]$ uporabimo v naslednji trditvi navedeno rekurzivno formulo, ki predstavlja predelavo splošne rekurzivne formule za razdaljo urejanja (4.3) v primer posplošene oz. utežene Levenshteinove razdalje. Rekurzija je zelo podobna že prej predstavljeni rekurziji Levenshteinove razdalje (4.4), samo da tu v posplošeni verziji namesto stroškov osnovnih operacij urejanja vrednosti 1 dovoljujemo tudi druge vrednosti. Kadar govorimo o grafu urejanja $G(s,t)$, te vrednosti imenujemo uteži $w(e) \in \mathbb{R}$. Uteži lahko predstavimo v obliki matrik operacij urejanja.

Dokaz trditve je podan v nadaljevanju.

Trditev 10. *Za $i = 0, 1, \ldots, m$–1 in $j = 0, 1, \ldots, n$–1, imamo*

$$\begin{aligned}
T[-1,-1] &= 0,\\
T[i,-1] &= T[i-1,-1] + Del(s[i]),\\
T[-1,j] &= T[-1,j-1] + Ins(t[j]),\\
T[i,j] &= \min \begin{cases} T[i-1,j-1) + Sub(s[i],t[j]),\\ T[i-1,j] + Del(s[i]),\\ T[i,j-1] + Ins(t[j]), \end{cases}
\end{aligned}$$



Slika 4.4: Prikaz rekurzije iz Trditve (10) oziroma prikaz odvisnosti vrednosti $T[i, j]$ od sosednjih treh pozicij. Slika predstavlja predelavo Slike 4.3 (rekurzija Levenshteinove razdalje) v primer posplošene Levenshteinove razdalje.

kjer $Sub(s[i], t[j])$ *predstavlja zamenjave, vključno z ujemanji, ko je* $s[i] = t[j]$. *Najpogosteje je strošek ujemanja enak nič, vendar je mogoče predpisati tudi drugače.*

Vrednost na poziciji $[i, j]$ v tabeli T, kjer sta $i, j \geq 0$, je odvisna samo od vrednosti na sosednih pozicijah $[i-1, j-1], [i-1, j]$ in $[i, j-1]$. Odvisnosti so prikazane na Sliki 4.4.

Algoritem GENERIC-DP (1), katerega koda je podana spodaj, izvaja izračun utežene Levenshteinove razdalje urejanja z uporabo tabele T (glej Tabelo 4.3). Iskana vrednost je $T[m-1, n-1] = ED(s, t)$ (posledica (1)).

Sedaj bomo dokazali veljavnost postopka izračuna algoritma, in sicer najprej z navedbo delnega rezultata.

Lema 1. *Za vsak* $a, b \in \Sigma$ *in* $u, v \in \Sigma^*$, *imamo*

$$\begin{aligned} ED(ua, \varepsilon) &= ED(u, \varepsilon) + Del(a), \\ ED(\varepsilon, vb) &= ED(\varepsilon, v) + Ins(b), \\ ED(ua, vb) &= \min \begin{cases} ED(u, v) + Sub(a, b), \\ ED(u, vb) + Del(a), \\ ED(ua, u) + Ins(b). \end{cases} \end{aligned}$$

Dokaz. Zaporedje operacij urejanja, ki transformirajo niz ua v prazen niz, lahko uredimo tako, da se konča z brisanjem znaka a. Preostali del

Algoritem 1 GENERIC-DP(s, m, t, n)

1: $T[-1,-1] \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** $m-1$ **do**
3: $\quad T[i,-1] \leftarrow T[i-1,-1] + Del(s[i])$
4: **end for**
5: **for** $j \leftarrow 0$ **to** $n-1$ **do**
6: $\quad T[-1,j] \leftarrow T[-1,j-1] + Ins(t[j])$
7: $\quad$ **for** $i \leftarrow 0$ **to** $m-1$ **do**
8: $$T[i,j] \leftarrow \min \begin{cases} T[i-1,j-1] + Sub(s[i],t[j]) \\ T[i-1,j] + Del(s[i]) \\ T[i,j-1] + Ins(t[j]) \end{cases}$$
9: $\quad$ **end for**
10: **end for**
11: **return** $T[m-1,n-1]$

zaporedja potem predstavlja pretvorbo niza u v prazen niz ε. Tako imamo:

$$\begin{aligned} ED(ua,\varepsilon) &= \min_{\sigma \in \mathcal{S}(ua,\varepsilon)} cost(\sigma) \\ &= \min_{\sigma' \in \mathcal{S}(u,\varepsilon)} cost(\sigma' \cdot (a,\varepsilon)) \\ &= \min_{\sigma' \in \mathcal{S}(u,\varepsilon)} cost(\sigma') + Del(a) \\ &= ED(u,\varepsilon) + Del(a). \end{aligned}$$

Torej drži prva identiteta. Veljavnost druge identitete lahko dokažemo po enakem postopku. Za tretjo zadošča, če ločimo primer, kjer je zadnja operacija urejanja zamenjava, brisanje ali vstavljanje. □

Dokaz. [Trditve (10)] Dokaz je direktna posledica enakosti $ED(\varepsilon,\varepsilon) = 0$ in leme (1), kjer določimo $a = s[i], b = t[j], u = s[0..i-1]$ in $v = t[0..j-1]$. □

Posledica 1. *Algoritem GENERIC-DP izračuna uteženo Levenshteinovo razdaljo urejanja med s in t.*

T	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	0	←1	2	3	4	5	6
0	T	1	1	↖1	2	3	4	5
1	A	2	2	2	↖2	3	4	5
2	C	3	3	3	2	↖2	←3	4
3	T	4	4	3	3	3	↖3	↖3
4	C	5	5	4	3	3	4	↖↑4

Tabela 4.3: Tabela T predstavlja matriko dinamičnega programiranja, ki jo dobimo pri izračunu posplošene Levenshteinove razdalje med nizoma TACTC in GTCCGT z algoritmom GENERIC-DP. Vrednosti stroškov operacij urejanja so $Sub(a, a) = 0$, $Sub(a, b) = 1$ za $a \neq b$ in $Del(a) = Ins(a) = 1$. Dobimo ED(TACTC, GTCCGT) $= T[4, 5] = 4$. Na tabeli lahko vidimo označeni dve poti minimalnega stroška, ki potekata od $[4, 5]$ do $[-1, -1]$ in predstavljata optimalni poravnavi. Izračunamo ju lahko z algoritmom ALIGNMENTS, ki bo podan v enem izmed naslednjih razdelkov:

$$\begin{pmatrix} - & \mathrm{T} & \mathrm{A} & \mathrm{C} & - & \mathrm{T} & \mathrm{C} \\ \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{C} & \mathrm{G} & \mathrm{T} & - \end{pmatrix} \text{ in } \begin{pmatrix} - & \mathrm{T} & \mathrm{A} & \mathrm{C} & \mathrm{T} & \mathrm{C} \\ \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{C} & \mathrm{G} & \mathrm{T} \end{pmatrix}.$$

Dokaz. Je posledica trditve (10), saj izračun, ki ga izvede algoritem, uporablja navedeno rekurzivno formulo. □

Medtem ko z direktnim programiranjem rekurzivne formule iz trditve (10) dobimo algoritem eksponentnega izvajalnega časa, lahko vidimo, da je čas izvajanja algoritma GENERIC-DP(s, m, t, n) kvadraten.

Trditev 11. *Algoritem GENERIC-DP, ki računa uteženo Levenshteinovo razdaljo med dvema nizoma, dolžin m in n, ima časovno zahtevnost $O(mn)$ in prostorsko zahtevnost $O(\min\{m, n\})$.*

Dokaz. Izračun vrednosti na vsaki poziciji v tabeli T je odvisen samo od treh sosednjih pozicij in vsak izračun se izvede v konstantnem času. Na ta

način se v tabeli T izračuna $m \times n$ vrednosti, po inicializaciji v času $O(m+n)$, kar nam da rezultat o izvajalnem času. Pri prostorski zahtevnosti moramo vedeti le, da je za izračun potreben prostor samo za dva stolpca ali vrstici tabele T. □

Podoben rezultat dobimo, če računamo vrednosti v tabeli T glede na antidiagonale. V tem primeru za pravilen izračun zadostuje, da si zapolnimo zadnje tri zaporedne antidiagonale.

4.3.2 Izračun optimalne poravnave

Algoritem GENERIC-DP (1) izračuna samo strošek preoblikovanja niza s v niz t. Da pa poleg tega dobimo še zaporedje operacij urejanja, ki pretvorijo s v t, ali pripadajočo poravnavo, moramo izvesti izračun s pregledom tabele T v obratnem vrstnem redu od pozicije $[m-1, n-1]$ do pozicije $[-1, -1]$. V poziciji $[i, j]$ med pregledanimi sosednjimi tremi pozicijami $[i-1, j-1], [i-1, j]$ in $[i, j-1]$ izberemo tisto, ki je generirala vrednost $T[i, j]$ (glej Tabelo 4.3). Implementacija te metode za izračun optimalne poravnave je predstavljena v algoritmu ONE-ALIGNMENT (2), katerega koda je navedena v nadaljevanju.

Veljavnost postopka lahko pojasnimo s pojmom *aktivne povezave* (angl. active arc) v grafu urejanja $G(s, t)$. To so povezave, ki določajo optimalno poravnavo.

Formalno zapišemo:

Definicija 43. *Povezava* $\big((i', j'), (i, j)\big)$*, označena* $label(a, b)$*, je* aktivna *natanko takrat, ko:*

$$T[i, j] = T[i', j'] + Sub(a, b), \text{ če } i' = i - 1 \text{ in } j' = j - 1,$$
$$T[i, j] = T[i', j'] + Del(a), \text{ če } i' = i - 1 \text{ in } j' = j,$$
$$T[i, j] = T[i', j'] + Ins(b), \text{ če } i' = i \text{ in } j' = j - 1,$$

kjer $i, i' \in \{-1, 0, \ldots, m-1\}$, $j, j' \in \{-1, 0, \ldots, n-1\}$ *in* $a, b \in \Sigma$.

$Sub(a, b)$ *tudi tu zajema tako operacije zamenjave kot tudi ujemanja.*

Lema 2. *Označena pot v grafu $G(s,t)$, ki povezuje (k,l) z (i,j), je optimalna poravnava nizov $s[k..i]$ in $t[l..j]$ natanko takrat, ko so vse njene povezave aktivne. Imamo:*

$$ED(s[k..i], t[l..j]) = T[i,j] - T[k,l].$$

Dokaz. Po definiciji (4.2) vemo, da je poravnava optimalna, če je strošek poti minimalen. Dokazati moramo:

$$ED(s[k..i], t[l..j]) = T[i,j] - T[k,l].$$

Dokažimo ekvivalenco z uporabo rekurzije glede na pozitivno dolžino poti, ki se šteje s številom povezav. Naj bo (i',j') vozlišče, ki se na poti pojavi pred vozliščem (i,j).

Najprej predpostavimo, da ima pot dolžino 1, to pomeni, da je $(k,l) = (i',j')$. Če je strošek poti minimalen, potem je njegova vrednost:

$$ED(s[k..i], t[l..j]) = T[i,j] - T[k,l].$$

In ker ta strošek, odvisno od obravnavanega primera, predstavlja eno izmed operacij $Sub(s[i], t[j])$, $Del(s[i])$ ali $Ins(t[j])$, lahko glede na definicijo (43) sklepamo, da je povezava aktivna.

Obratno imamo, če je povezava v poti aktivna, po definiciji (43) bodisi $T[i,j] - T[k,l] = Sub(s[i], t[j])$, $T[i,j] - T[k,l] = Del(s[i])$ ali $T[i,j] - T[k,l] = Ins(t[j])$, glede na obravnavan primer. Ampak te vrednosti so tudi razdalje med dvema nizoma, ki imata dolžino le 1. Torej je strošek poti minimalen.

Sedaj predpostavimo, da je pot večja od 1.

Če je strošek poti minimalen, velja enako za njen odsek, ki povezuje (k,l) z (i',j'), in za povezavo $\big((i',j'),(i,j)\big)$. Če na prvem odseku uporabimo rekurzijo, dobimo rezultat, da je odsek sestavljen iz aktivnih povezav. Iz minimalnosti stroška zadnje povezave sledi, da je tudi ta povezava aktivna (glej trditev (10)).

V drugo smer predpostavimo, da so vse povezave v poti aktivne. Z uporabo rekurzije na odseku poti od (k,l) do (i',j') sklepamo, da je njen strošek

minimalen in:

$$T[i', j'] - T[k, l] = ED(s[k..i'], t[l..j']).$$

Ker je tudi zadnja povezava aktivna, je glede na trditev (10) njen strošek minimalen in je enak $T[i, j]-T[i', j']$. Torej je strošek celotne poti minimalen:

$$\begin{aligned} ED(s[k..i], t[l..j]) &= (T[i, j] - T[i', j']) + (T[i', j'] - T[k, l]) \\ &= T[i, j] - T[k, l]. \end{aligned}$$

□

Opazimo, da v vsako vozlišče grafa urejanja, razen $(-1, -1)$, vstopi vsaj ena aktivna povezava (glede na rekurzivno formulo v trditvi (10)). Delo, ki ga opravi algoritem ONE-ALIGNMENT, je torej sestavljeno iz pregleda grafa urejanja navzgor po aktivnih povezavah in ustavitve, ko doseže vozlišče $(-1, -1)$ (glej Tabelo 4.3). V algoritmu spremenljivka z predstavlja niz nad abecedo $\big(\Sigma \cup \{-\}\big) \times \big(\Sigma \cup \{-\}\big) \setminus \{(-, -)\}$ in se tvori s postopno konkatenacijo posameznih komponent.

Trditev 12. *Izvršitev ONE-ALIGNMENT(s, m, t, n) nam da optimalno poravnavo nizov s in t, to je poravnava s stroškom $ED(s, t)$. Izračun se izvede v času in dodatnem prostoru $O(m + n)$.*

Dokaz. Formalni dokaz temelji na lemi (2). Opazimo, da pogoja v vrsticah 4 in 7 testirata aktivnost povezav v grafu urejanja. Tretji primer, obravnavan v vrsticah $10-13$, ustreza tretjemu pogoju iz definicije o aktivnih povezavah, saj v vsako vozlišče v grafu, različno od $(-1, -1)$, vedno vstopi vsaj ena aktivna povezava. Celoten izračun nam tako da označeno pot iz izhodišča $(-1, -1)$ do konca $(m-1, n-1)$, ki je sestavljena samo iz aktivnih povezav. Glede na lemo (2) je ta pot optimalna poravnava nizov s in t.

Vsaka operacija, ki pomembno vpliva na čas izvajanja algoritma, povzroči zmanjšanje vrednosti i ali vrednosti j, ki se spreminjata od $m - 1$ in $n - 1$ postopoma do -1. To nam da izvajalni čas $O(m + n)$. Potreben je tudi dodaten prostor za hrambo niza z, katerega maksimalna dolžina je $m + n$. S tem je dokaz potrjen. □

Algoritem 2 ONE-ALIGNMENT(s, m, t, n)

1: $z \leftarrow (\varepsilon, \varepsilon)$
2: $(i, j) \leftarrow (m-1, n-1)$
3: **while** $i \neq -1$ **and** $j \neq -1$ **do**
4: **if** $T[i,j] = T[i-1, j-1] + Sub(s[i], t[j])$ **then**
5: $z \leftarrow (s[i], t[j]) \cdot z$
6: $(i, j) \leftarrow (i-1, j-1)$
7: **else if** $T[i,j] = T[i-1, j] + Del(s[i])$ **then**
8: $z \leftarrow (s[i], \text{“}-\text{”}) \cdot z$
9: $i \leftarrow i - 1$
10: **else**
11: $z \leftarrow (\text{“}-\text{”}, t[j]) \cdot z$
12: $j \leftarrow j - 1$
13: **end if**
14: **end while**
15: **while** $i \neq -1$ **do**
16: $z \leftarrow (s[i], \text{“}-\text{”}) \cdot z$
17: $i \leftarrow i - 1$
18: **end while**
19: **while** $j \neq -1$ **do**
20: $z \leftarrow (\text{“}-\text{”}, t[j]) \cdot z$
21: $j \leftarrow j - 1$
22: **end while**
23: **return** z

Naj omenimo, da lahko test preverjanja, katera izmed treh povezav pride v vozlišče (i, j) v grafu urejanja, izvedemo v kateremkoli vrstnem redu. Obstaja namreč $3! = 6$ možnih zapisov vrstic $4 - 13$. Predstavljena verzija daje prednost poti, ki vsebuje diagonalne povezave. Na primer, če zamenjamo vrstice $4 - 6$ z vrsticami $7 - 9$, dobimo najvišjo pot. Rezultat lahko sprogramiramo tudi tako, da dobimo naključno izmed optimalnih poravnav.

Za izračun poravnave je tudi mogoče hraniti aktivne povezave kot povratne povezave (angl. return arcs) v dodatni tabeli med računanjem vrednosti tabele T. Izračun poravnave potem preberemo iz tabele povratnih povezav tako, da sledimo tem povezavam od pozicije $[m-1, n-1]$ do pozicije $[-1, -1]$. Za to potrebujemo $O(mn)$ prostora, enako kot za tabelo T. Za vsako pozicijo je dovolj, če hranimo le po eno povratno povezavo izmed treh možnih, kar je lahko zakodirano le z dvema bitoma.

Postopek za izračun optimalne poravnave, predstavljen v tem razdelku, uporablja tabelo T in zahteva kvadraten prostor, vendar je mogoče najti optimalno poravnavo tudi v linearnem prostoru z uporabo metode deli in vladaj opisane v razdelku o najdaljšem skupnem podzaporedju.

4.3.3 Izračun vseh optimalnih poravnav

Če hočemo vedeti vse možne optimalne poravnave nizov s in t, lahko uporabimo algoritem ALIGNMENTS (3), katerega koda je podana spodaj. Algoritem kliče proceduro AL, kjer se predpostavlja, da so spremenljivke s, t in T globalne. Tako kot prejšnji algoritem tudi ta temelji na ideji aktivnih povezav (glej Tabelo 4.3).

Trditev 13. *Algoritem ALIGNMENTS ustvari vse optimalne poravnave vhodnih nizov. Njegov izvajalni čas je sorazmeren z vsoto dolžin vseh ustvarjenih poravnav.*

Dokaz. Opazimo, da s testi v vrsticah $1, 4$ in 7 preverimo aktivnost povezav. S testom v vrstici 10 ustvarimo trenutno poravnavo, ko je ta zaključena. Preostali del dokaza je podoben tistemu za algoritem ONE-ALIGNMENT.

Izvajalni čas posameznega testa je konstanten. Poleg tega se z vsakim testom trenutna poravnava poveča za en par. Torej drži rezultat za skupni izvajalni čas. □

Algoritem 3 ALIGNMENTS(s, m, t, n)

1: AL$(m-1, n-1, (\varepsilon, \varepsilon))$

Procedura 4 AL(i, j, z)

1: **if** $i \neq -1$ **and** $j \neq -1$ **and** $T[i,j] = T[i-1, j-1] + Sub(s[i], t[j])$ **then**
2: AL$(i-1, j-1, (s[i], t[j]) \cdot z)$
3: **end if**
4: **if** $i \neq -1$ **and** $T[i,j] = T[i-1, j] + Del(s[i])$ **then**
5: AL$(i-1, j, (s[i], \text{“}-\text{”}) \cdot z)$
6: **end if**
7: **if** $j \neq -1$ **and** $T[i,j] = T[i, j-1] + Ins(t[j])$ **then**
8: AL$(i, j-1, (\text{“}-\text{”}, t[j]) \cdot z)$
9: **end if**
10: **if** $i = -1$ **and** $j = -1$ **then**
11: sporočilo, da je z poravnava
12: **end if**

Tudi pri izračunu vseh poravnav si lahko zapomnimo povratne povezave, tako kot zgoraj, vendar je tu potrebno hraniti tudi do tri povezave za posamezno pozicijo, kar lahko zakodiramo s tremi biti.

Ni pa dobro izračunati vseh poravnav, če je teh preveč (glej trditev (7)). Bolj primerno je ustvariti graf, ki vsebuje vse te informacije, in ga lahko poizvedujemo tudi kasneje.

4.4 Najdaljše skupno podzaporedje

V tem razdelku nas zanima izračun najdaljšega podzaporedja, ki je skupno dvema nizoma [CHL01]. Najprej definirajmo osnovne pojme:

Definicija 44 (Skupno podzaporedje)**.** *w je* skupno podzaporedje *(angl. common subsequence) od s in t, če je w podzaporedje tako od s kot od t.*

Definicija 45 (Najdaljše skupno podzaporedje)**.** Najdaljše skupno podzaporedje *$Lcs(s,t)$ (angl. longest common subsequence) dveh nizov je skupno pozdaporedje maksimalne dolžine.*

Problem 5 (Najdaljše skupno podzaporedje)**.**
Naloga: Končna abeceda Σ, niza s in t iz Σ^*.
Dopustna rešitev: Niz $w \in \Sigma^*$ tako, da je w skupno podzaporedje od s in t; tj. w dobimo z odvzemom znakov iz niza s ali niza t.
Mera/ciljna funkcija: Dolžina podzaporedja, $|w|$.
Cilj: Maksimizacija.

Ta problem je poseben primer razdalje urejanja, kjer se ne upoštevajo zamenjave, ampak so dovoljena samo vstavljanja in brisanja. Dva niza s in t lahko imata več najdaljših skupnih podzaporedij. Množica teh nizov se označi $Lcs(s,t)$. Enotno dolžino nizov iz $Lcs(s,t)$ pa označimo $lcs(s,t)$.

Če določimo

$$Sub(a,a) = 0 \tag{4.7}$$

in

$$Del(a) = Ins(a) = 1 \tag{4.8}$$

za $a \in \Sigma$, in če predpostavimo

$$Sub(a,b) > Del(a) + Ins(b) = 2 \tag{4.9}$$

za $a, b \in \Sigma$ in $a \neq b$, vrednost $T[m-1, n-1]$ (glej trditev (10)) predstavlja *razdaljo podzaporedja* (angl. subsequence distance) oz. *Indel razdaljo* med s in t, z oznako $Indel(s,t)$ (glej Tabelo 4.4). Njen izračun je dualen problemu izračuna dolžine najdaljšega skupnega podzaporedja (angl. longest common subsequences) nizov s in t zaradi sledeče trditve.

Trditev 14. *Razdalja podzaporedja izpolnjuje naslednjo enakost:*

$$Indel(s,t) = |s| + |t| - 2 \times lcs(s,t).$$

T	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	0	$\leftarrow 1$	2	3	4	5	6
0	T	1	2	$\nwarrow 1$	$\leftarrow 2$	3	4	5
1	A	2	3	$\uparrow 2$	$\leftarrow\uparrow 3$	4	5	6
2	C	3	4	3	$\nwarrow 2$	$\leftarrow\nwarrow 3$	$\leftarrow 4$	5
3	T	4	5	4	$\uparrow 3$	4	5	$\nwarrow 4$
4	C	5	6	5	4	$\nwarrow 3$	$\leftarrow 4$	$\leftarrow\uparrow 5$

Tabela 4.4: Primer izračuna Indel razdalje. Tabela T predstavlja matriko dinamičnega programiranja, ki jo dobimo z algoritmom GENERIC-DP (1). Primer je enak tistemu iz Tabele 4.3, samo da smo tu spremenili vrednosti stroškov operacij urejanja, ki so sedaj $Sub(a, a) = 0$, $Sub(a, b) = 3$ za $a \neq b$ in $Del(a) = Ins(a) = 1$. Dobimo $Indel$(TACTC, GTCCGT) $= T[4, 5] = 5$. V tabeli T so označene spodaj naštete pripadajoče optimalne poravnave, izračunane z algoritmom ALIGNMENTS (3) (maksimalne poti od $[4, 5]$ do $[-1, -1]$). Lahko opazimo, da nam prve tri poravnave dajo najdaljše skupno podzaporedje TCT, zadnja poravnava pa TCC. Indel razdaljo lahko tudi izračunamo $|\text{TACTC}| + |\text{GTCCGT}| - 2 \times |\text{TCC}| = 5$.

$$\begin{pmatrix} - & \mathbf{T} & - & \text{A} & \mathbf{C} & - & \mathbf{T} & \text{C} \\ \text{G} & \mathbf{T} & \text{C} & - & \mathbf{C} & \text{G} & \mathbf{T} & - \end{pmatrix}, \begin{pmatrix} - & \mathbf{T} & \text{A} & \mathbf{C} & - & - & \mathbf{T} & \text{C} \\ \text{G} & \mathbf{T} & - & \mathbf{C} & \text{C} & \text{G} & \mathbf{T} & - \end{pmatrix},$$

$$\begin{pmatrix} - & \mathbf{T} & \text{A} & - & \mathbf{C} & - & \mathbf{T} & \text{C} \\ \text{G} & \mathbf{T} & - & \text{C} & \mathbf{C} & \text{G} & \mathbf{T} & - \end{pmatrix}, \begin{pmatrix} - & \mathbf{T} & \text{A} & \mathbf{C} & \text{T} & \mathbf{C} & - & - \\ \text{G} & \mathbf{T} & - & \mathbf{C} & - & \mathbf{C} & \text{G} & \text{T} \end{pmatrix}.$$

Dokaz. Po definiciji je $Indel(s, t)$ minimalen strošek poravnave dveh nizov, izračunan iz osnovnih stroškov Sub, Del in Ins, ki zadovoljujejo zgornje predpostavke (4.7), (4.8), (4.9). Naj bo $(\tilde{s}, \tilde{t})$ poravnava s stroškom $Indel(s, t)$. Neenakost:

$$Sub(a, b) > Del(a) + Ins(b)$$

pomeni, da $(\tilde{s}, \tilde{t})$ ne vsebuje nobene zamenjave dveh različnih znakov, saj brisanje a in vstavljanje b znaša manj kot zamenjava a z b, ko $a \neq b$. Zaradi $Del(a) = Ins(b) = 1$ je vrednost $Indel(s,t)$ enaka številu vstavljanj in brisanj v poravnavi $(\tilde{s}, \tilde{t})$. Ostali poravnani pari v $(\tilde{s}, \tilde{t})$ ustrezajo ujemanjem in njihovo število je $lcs(s,t)$ (ne more biti manjše, drugače bi bilo v protislovju z definicijo $Indel(s,t)$). Če vsak tak par zamenjamo z vstavljanjem in takojšnjim brisanjem istega znaka, dobimo poravnavo, ki vsebuje samo vstavljanja in brisanja; njena dolžina je $|s| + |t|$. Strošek poravnave $(\tilde{s}, \tilde{t})$ je torej $|s| + |t| - 2 \times lcs(s,t)$, kar nam potrdi zgornjo enakost. □

Naivna metoda za izračun $lcs(s,t)$ upošteva vsa podzaporedja od s in jih primerja s podzaporedji od t ter obdrži najdaljše. Niz s, dolžine m, nam lahko da 2^m različnih podzaporedij, tako da je ta metoda preštevanja neuporabna za večje vrednosti m.

4.4.1 Izračun z dinamičnim programiranjem

Z uporabo metode dinamičnega programiranja, ki je zelo podobna izračunu utežene Levenshteinove razdalje iz prejšnjega razdelka, je mogoče izračunati $Lcs(s,t)$ in $lcs(s,t)$ v času in prostoru $O(mn)$. Metoda postopoma izračunava dolžine najdaljših skupnih podzaporedij med vedno daljšimi predponami dveh nizov s in t.

Za ta namen ustvarimo dvodimenzionalno tabelo S z $m+1$ vrsticami in $n+1$ stolpci, ki je definirana za $i = -1, 0, \ldots, m-1$ in $j = -1, 0, \ldots, n-1$ takole:

$$S[i,j] = \begin{cases} 0 & \text{če } i = -1 \text{ ali } j = -1, \\ lcs(s[0..i], t[0..j]) & \text{sicer.} \end{cases}$$

Izračun

$$lcs(s,t) = S[m-1, n-1]$$

temelji na rekurzivni relaciji iz naslednje trditve.

Trditev 15. *Za* $i = 0, 1, \ldots, m-1$ *in* $j = 0, 1, \ldots, n-1$, *imamo*

$$S[i,j] = \begin{cases} S[i-1, j-1] + 1 & \text{če } s[i] = t[j], \\ \max\{S[i-1, j], S[i, j-1]\} & \text{sicer.} \end{cases}$$

Dokaz. Naj bo $ua = s[0..i]$ in $vb = t[0..j]$ $(u, v \in \Sigma^*,\ a, b \in \Sigma)$. Če je $a = b$, se najdaljše skupno podzaporedje med ua in vb nujno konča z a. Posledično je oblike wa, kjer je w najdaljše podzaporedje skupno nizoma u in v. Torej je v tem primeru $S[i,j] = S[i-1, j-1] + 1$.

Če $a \neq b$ ter če ua in vb vsebuje najdaljše skupno podzaporedje, ki se ne konča z a, imamo $S[i,j] = S[i-1, j]$. Simetrično, če se ne konča z b, imamo $S[i,j] = S[i, j-1]$. To se pravi $S[i,j] = \max\{S[i-1, j], S[i, j-1]\}$. □

Enakost, podana v zadnji trditvi, je v algoritmu LCS-SIMPLE (5) uporabljena za izračun vseh vrednosti tabele S in vrednosti $lcs(s,t) = S[m-1, n-1]$.

Algoritem 5 LCS-SIMPLE(s, m, t, n)

```
1:  for i ← −1 to m − 1 do
2:      S[i, −1] ← 0
3:  end for
4:  for j ← 0 to n − 1 do
5:      S[−1, j] ← 0
6:      for i ← 0 to m − 1 do
7:          if s[i] = t[j] then
8:              S[i, j] ← S[i − 1, j − 1] + 1
9:          else
10:             S[i, j] ← max{S[i − 1, j], S[i, j − 1]}
11:         end if
12:     end for
13: end for
14: return S[m − 1, n − 1]
```

Tabela 4.5 prikazuje delovanje algoritma.

S	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	0	$\leftarrow 0$	0	0	0	0	0
0	T	0	0	$\nwarrow 1$	$\leftarrow 1$	1	1	1
1	A	0	0	$\uparrow 1$	$\leftarrow \uparrow 1$	1	1	1
2	C	0	0	1	$\nwarrow 2$	$\leftarrow \nwarrow 2$	$\leftarrow 2$	2
3	T	0	0	1	$\uparrow 2$	2	2	$\nwarrow 3$
4	C	0	0	1	2	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow \uparrow 3$

Tabela 4.5: Primer izračuna najdaljšega skupnega podzaporedja $lcs(s,t)$ nizov s = TACTC in t = GTCCGT. Z Algoritmom LCS-SIMPLE dobimo dolžino lcs(TACTC, GTCCGT) = $S[4,5] = 3$. V tabeli S so predstavljene tudi štiri označene poti maksimalnega stroška med pozicijama $[-1,-1]$ in $[4,5]$. Spodnje štiri pripadajoče poravnave (enake kot v Tabeli 4.4) potrdijo, da je Lcs(TACTC, GTCCGT) = {TCT, TCC}:

$$\begin{pmatrix} - & \mathbf{T} & - & \mathrm{A} & \mathbf{C} & - & \mathbf{T} & \mathrm{C} \\ \mathrm{G} & \mathbf{T} & \mathrm{C} & - & \mathbf{C} & \mathrm{G} & \mathbf{T} & - \end{pmatrix}, \begin{pmatrix} - & \mathbf{T} & \mathrm{A} & \mathbf{C} & - & - & \mathbf{T} & \mathrm{C} \\ \mathrm{G} & \mathbf{T} & - & \mathbf{C} & \mathrm{C} & \mathrm{G} & \mathbf{T} & - \end{pmatrix},$$

$$\begin{pmatrix} - & \mathbf{T} & \mathrm{A} & - & \mathbf{C} & - & \mathbf{T} & \mathrm{C} \\ \mathrm{G} & \mathbf{T} & - & \mathrm{C} & \mathbf{C} & \mathrm{G} & \mathbf{T} & - \end{pmatrix}, \begin{pmatrix} - & \mathbf{T} & \mathrm{A} & \mathbf{C} & \mathrm{T} & \mathbf{C} & - & - \\ \mathrm{G} & \mathbf{T} & - & \mathbf{C} & - & \mathbf{C} & \mathrm{G} & \mathrm{T} \end{pmatrix}.$$

Trditev 16. *Algoritem LCS-SIMPLE izračuna maksimalno dolžino podzaporedij, ki so skupni s in t. Izvajalni čas in prostor sta $O(mn)$.*

Dokaz. Pravilnost algoritma izhaja iz rekurzivne formule v trditvi (15). Očitno je, da sta izvajalni čas in prostor enaka $O(mn)$. □

Tako kot pri izračunu optimalne poravnave v prejšnjem razdelku lahko tu na podoben način najdemo najdaljše skupno podzaporedje nizov s in t. Dobimo ga s pregledom izračunane tabele S v obratnem vrstnem redu od pozicije $[m-1, n-1]$ nazaj. Koda, ki sledi (6), izvaja ta izračun na enak

način kot algoritem ONE-ALIGNMENT (2).

Algoritem 6 ONE-LCS(s, m, t, n, S)

```
1: z ← ε
2: (i, j) ← (m − 1, n − 1)
3: while i ≠ −1 and j ≠ −1 do
4:    if s[i] = t[j] then
5:       z ← s[i] · z
6:       (i, j) ← (i − 1, j − 1)
7:    else if S[i − 1, j] > S[i, j − 1] then
8:       i ← i − 1
9:    else
10:      j ← j − 1
11:   end if
12: end while
13: return z
```

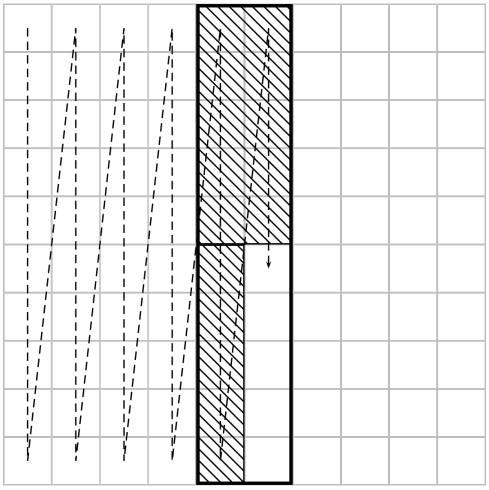
Mogoče je izračunati tudi vsa najdaljša podzaporedja, skupna nizom s in t, kot v prejšnjem razdelku, z razširitvijo tehnike, uporabljene v algoritmu ONE-LCS.

4.4.2 Izračun dolžine v linearnem prostoru

Če nas zanima samo dolžina najdaljšega skupnega podzaporedja, je za izračun dovolj, če si zapomnimo samo dva stolpca (ali dve vrstici) v tabeli S (Slika 4.5). Takšna ideja je realizirana v spodnjem algoritmu LCS-COLUMN (7).

Trditev 17. *Izvajanje algoritma LCS-COLUMN(s, m, t, n) nam da tabelo C, katere vrednost $C[i]$, za $i = -1, 0, \ldots, m-1$, je enaka $lcs(s[0..i], t)$. Izračun porabi $O(mn)$ časa in $O(m)$ prostora.*

Dokaz. Izvajanje algoritma nam da tabelo C_1. Glede na različne vrednosti j moramo dokazati, da je $C_1[i] = S[i, j]$, za $i = -1, 0, \ldots, m-1$. Namreč



Slika 4.5: Potek algoritma LCS-COLUMN, prikazan z dinamično matriko. Izračun dolžine $lcs(s,t)$ ne zahteva več kot $2m$ prostora za problem velikosti $m \times n$. Za izračun vrednosti v posameznem stolpcu potrebujemo samo vrednosti iz predhodnega stolpca. Vir slike: [JP04].

Algoritem 7 LCS-COLUMN(s, m, t, n)

```
1:  for i ← −1 to m − 1 do
2:      C1[i] ← 0
3:  end for
4:  for j ← 0 to n − 1 do
5:      C2[−1] ← 0
6:      for i ← 0 to m − 1 do
7:          if s[i] = t[j] then
8:              C2[i] ← C1[i − 1] + 1
9:          else
10:             C2[i] ← max{C1[i], C2[i − 1]}
11:         end if
12:     end for
13:     C1 ← C2
14: end for
15: return C1
```

na koncu izvajanja algoritma, ko je $j = n - 1$, dobimo $C_1[i] = S[i, n-1] = lcs(s[0..i], t)$, za $i = -1, 0, \ldots, m-1$, po definiciji tabele S.

Pred izvajanjem zanke v vrsticah $4 - 14$ obdelamo primer, ko je $j = -1$; imamo $C_1[i] = 0$ za vsak i. Imamo tudi $S[i, -1] = 0$, kar dokazuje, da relacija drži za $j = -1$.

Obravnavajmo sedaj primer, ko ima j nenegativno vrednost. Ustrezne vrednosti tabele C_1 se izračunajo v vrsticah $5 - 13$ algoritma. Glede na ukaz v vrstici 13 je dovolj, če pokažemo, da tabela C_2 izpolnjuje zgornjo relacijo takrat, ko jo izpolnjuje C_1 za vrednost $j - 1$. Torej predpostavimo, da je $C_1[i] = S[i, j-1]$, za $i = -1, 0, \ldots, m-1$ in pokažemo, da po izvajanju vrstic $5 - 12$ dobimo $C_2[i] = S[i, j]$ za $i = -1, 0, \ldots, m-1$.

Dokaz temelji na različnih vrednostih i. Za $i = -1$, to ustreza inicializaciji tabele C_2 v vrstici 5, dobimo $C_2[-1] = 0 = S[-1, j]$. Ko je $i \geq 0$, moramo upoštevati dva primera. Če je $s[i] = t[j]$, ustrezen ukaz nastavi $C_2[i] = C_1[i-1] + 1$, kar je enako $S[i-1, j-1] + 1$. Ta vrednost je tudi $S[i, j]$ po trditvi (15), kar nam da na koncu $C_2[i] = S[i, j]$. Če $s[i] \neq t[j]$, nam ukaz v vrsticah $9 - 10$ da $C_2[i] = \max\{C_1[i], C_2[i-1]\}$. To je enako $\max\{S[i, j-1], C_2[i-1]\}$, po tem ko upoštevamo vrednosti j, in $\max\{S[i, j-1], S[i-1, j]\}$, ko upoštevamo vrednosti i. Končno dobimo iskani rezultat $C_2[i] = S[i, j]$, ponovno po trditvi (15).

Tako se zaključi pregled vseh različnih vrednosti i in j ter s tem dobimo rezultat. □

Algoritma LCS-COLUMN za izračun maksimalne dolžine skupnih podzaporedij nizov s in t ne moremo kar tako enostavno pretvoriti v algoritem za izračun najdaljšega skupnega podzaporedja, kot smo to storili prej (ker ne hranimo vrednosti celotne tabele S). Vendar lahko algoritem uporabimo pri vmesnih izračunih v metodi, ki sledi v naslednjem razdelku.

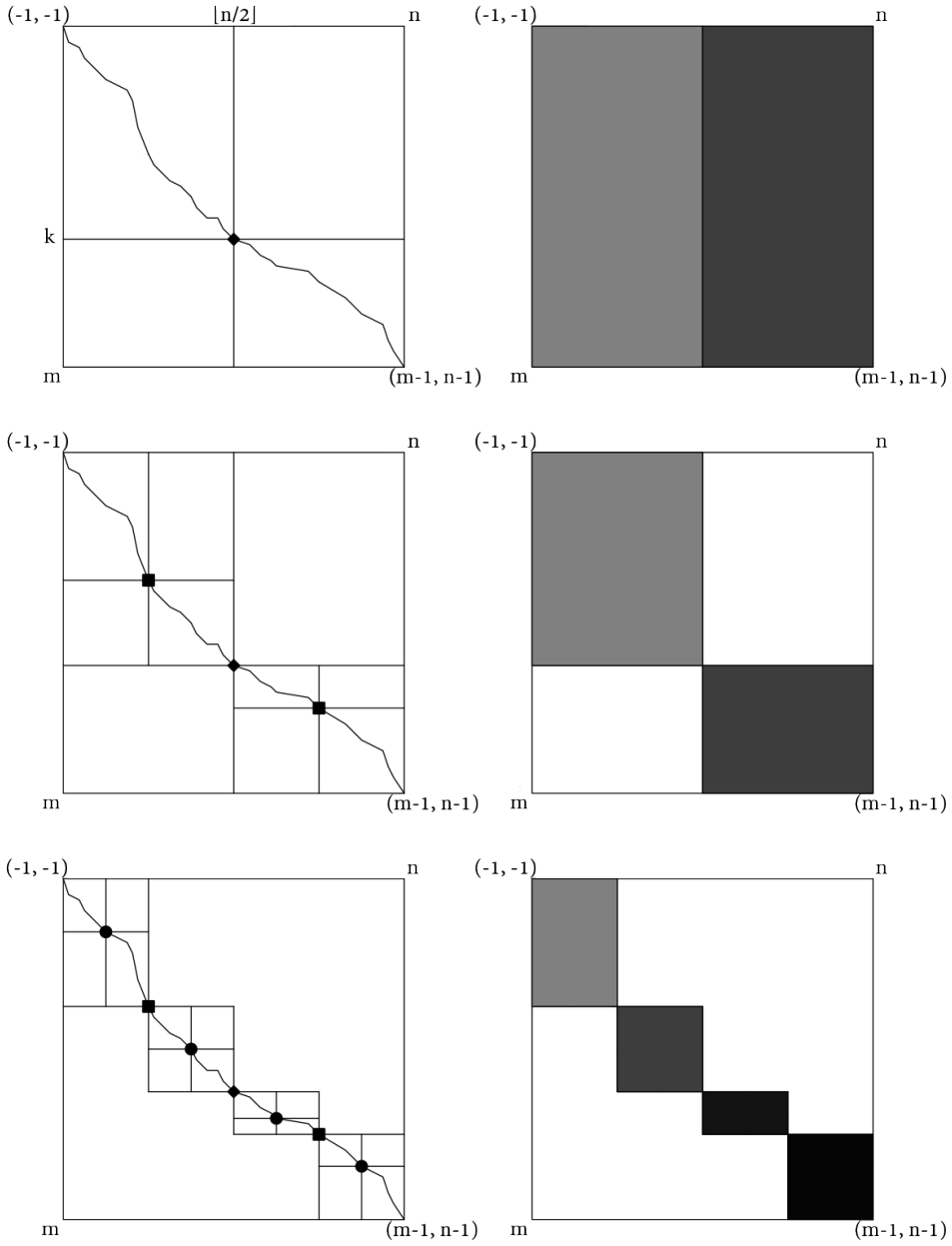
4.4.3 Izračun najdaljšega skupnega podzaporedja v linearnem prostoru

Zdaj bomo pokazali, kako predstaviti najdaljše skupno podzaporedje s pomočjo metode deli in vladaj [Hir75]. Metoda se v celoti izvrši v linearnem prostoru na račun podvojitve izvajalnega časa. Idejo izračuna lahko opišemo z ustreznim grafom urejanja nizov s in t. Sestavljena je iz določitve vozlišča oblike $(k-1, \lfloor n/2 \rfloor - 1)$, $0 \leq k \leq m$, skozi katerega gre pot maksimalnega stroška iz $(-1,-1)$ v $(m-1, n-1)$ v grafu $G(s,t)$. Bistveno je, da lahko dolžino te poti učinkovito izračunamo, ne da bi pot poznali. Ko je to vozlišče znano, moramo samo še izračunati s tem vozliščem določena odseka poti, od $(-1,-1)$ do $(k-1, \lfloor n/2 \rfloor - 1)$, in od $(k-1, \lfloor n/2 \rfloor - 1)$ do $(m-1, n-1)$. Ta izračun nam da najdaljše podzaporedje u, ki je skupno podnizoma $s[0..k-1]$ in $t[0..\lfloor n/2 \rfloor - 1]$, ter najdaljše podzaporedje v, ki je skupno podnizoma $s[k..m-1]$ in $t[\lfloor n/2 \rfloor..n-1]$. Ta dva izračuna se nato izvedeta dalje z rekurzivno uporabo iste metode (glej Sliko 4.6). Niz $z = u \cdot v$ je torej najdaljše skupno podzaporedje med s in t. Rekurzivni klic se ustavi, ko je eden izmed dveh nizov prazen ali vsebuje samo en znak.

Ostane nam še opis, kako izberemo vrednost k, ki določa vozlišče $(k-1, \lfloor n/2 \rfloor - 1)$. Celo število k je po definiciji indeks med 0 in m, za katerega je vrednost

$$lcs\Big(s\big[0..k-1\big], t\big[0..\lfloor n/2 \rfloor - 1\big]\Big) + lcs\Big(s\big[k..m-1\big], t\big[\lfloor n/2 \rfloor..n-1\big]\Big)$$

največja. Da ga najdemo, algoritem LCS (8) (koda je v nadaljevanju) v vrstici 8 začne z izračunom stolpca, indeksa $\lfloor n/2 \rfloor - 1$, tabele S s klicem algoritma LCS-COLUMN(s, m, t, $\lfloor n/2 \rfloor$) (7). Nato nadaljuje ta korak (vrstice $9-24$) z obdelavo druge polovice tabele S, kot to za prvo polovico stori algoritem LCS-COLUMN, vendar poleg tega shrani tudi kazalce na sredinski stolpec. Algoritem uporablja dve tabeli C_1 in C_2, zato da lahko izračuna vrednosti iz S, in dve dodatni tabeli P_1 in P_2 za hrambo kazalcev.



Slika 4.6: Prikaz metode deli in vladaj za izračun najdaljšega skupnega podzaporedja $Lcs(s,t)$ v linearnem prostoru. Izvajalni čas (površina pobarvanih pravokotnikov) se z vsako iteracijo razpolovi in posledično dobimo skupni izvajalni čas $O(mn)$. Vir slike: [JP04].

Zadnji dve tabeli sta implementacija tabele P, ki je definirana za $j = \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor, \ldots, n-1$ in $i = -1, 0, \ldots, m-1$:

$$P[i, j] = k$$

natanko takrat, ko je

$$0 \leq k \leq i + 1$$

in

$$\begin{aligned} lcs\Big(s\big[0..i\big], t\big[0..j\big]\Big) = \\ lcs\Big(s\big[0..k-1\big], t\big[0..\lfloor n/2 \rfloor - 1\big]\Big) \\ + lcs\Big(s\big[k..i\big], t\big[\lfloor n/2 \rfloor..j\big]\Big). \end{aligned} \tag{4.10}$$

Sledeča trditev je osnova, ki jo uporablja algoritem LCS za izračun vrednosti v tabeli P. Opazimo, da navedena rekurzija omogoča računanje stolpca po stolpcu, podobno kot pri računanju tabele S v algoritmu LCS-COLUMN.

Trditev 18. *Tabela P zadošča naslednji rekurzivni formuli:*

$$P\big[i, \lfloor n/2 \rfloor - 1\big] = i + 1$$

za $i = -1, 0, \ldots, m-1$,

$$P[-1, j] = 0$$

za $j \geq \lfloor n/2 \rfloor$, *in*

$$P[i, j] = \begin{cases} P[i-1, j-1] & \text{če } s[i] = t[j], \\ P[i-1, j] & \text{če } s[i] \neq t[j] \text{ in } S[i-1, j] \geq S[i, j-1], \\ P[i, j-1] & \text{sicer,} \end{cases}$$

za $i = 0, 1, \ldots, m-1$ *in* $j = \lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \ldots, n-1$.

P_1	j		3	4	5
i		$t[j]$	C	G	T
-1	$s[i]$	0	0	0	0
0	T	1	1	1	0
1	A	2	1	1	0
2	C	3	2	2	2
3	T	4	2	2	2
4	C	5	4	4	2

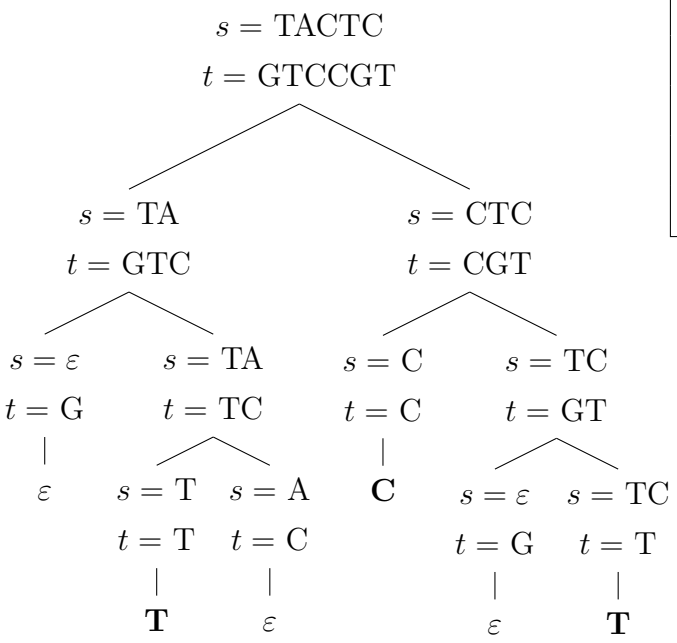


Tabela in drevo 4.6: Prikaz izvajanja algoritma LCS za izračun najdaljšega skupnega podzaporedja nizov s = TACTC in t = GTCCGT v linearnem prostoru. Zgoraj lahko vidimo drevo rekurzivnih klicev, kjer so prikazane delitve nizov po metodi deli in vladaj. Najdaljše skupno podzaporedje $Lcs(s,t)$ = TCT dobimo s konkatenacijo rezultatov, dobljenih z listov drevesa od leve proti desni. Zraven drevesa je prikazan primer izračuna vrednosti kazalcev tabele P_1, ki se zgodi pri prvem postopku delitve nizov s, t. Prvi stolpec predstavlja inicializacijo, ki se izvede v vrsticah $9-11$, temu pa sledijo stolpci, ki prikazujejo vrednosti za različne j po vsaki iteraciji **for** zanke v vrsticah $12-24$. Vrednost k je dobljena v koraku $j = 5$, ko je $k = P_1[4] = 2$, in to ustreza delitvi Lcs(TACTC, GTCCGT) = Lcs(TA, GTC) + Lcs(CTC, CGT).

Dokaz. Dokažimo trditev s pomočjo rekurzije glede na par (i, j).

Če je $j = \lfloor n/2 \rfloor - 1$, iz definicije P sledi, da je $k = i + 1$, saj je drugi del vsote v enačbi (4.10) enak nič zaradi tega, ker je $t[\lfloor n/2 \rfloor..j]$ prazen niz. Inicializacija rekurzije je tako pravilna.

Zdaj poglejmo primer, ko je $j \geq \lfloor n/2 \rfloor$. Če je $i = -1$, iz definicije P sledi, da je $k = 0$ in enačbi (4.10) je trivialno zadoščeno, saj so obravnavani členi podniza s prazni.

Preostane nam še splošni primer, ko je $j \geq \lfloor n/2 \rfloor$ in $i \geq 0$. Predpostavimo, da je $s[i] = t[j]$. Torej v grafu urejanja obstaja pot maksimalnega stroška, od $(-1, -1)$ do (i, j), ki gre skozi vozlišče $(i-1, j-1)$. Torej obstaja pot maksimalnega stroška, ki gre skozi $(k-1, \lfloor n/2 \rfloor - 1)$, kjer je $k = P[i-1, j-1]$. Z drugimi besedami na podlagi trditve (15) dobimo $lcs(s[0..i], t[0..j]) = lcs(s[0..i-1], t[0..j-1]) + 1$ in zaradi rekurzije (4.10) sledi $lcs(s[0..i-1], t[0..j-1]) = lcs(s[0..k-1], t[0..\lfloor n/2 \rfloor - 1]) + lcs(s[k..i-1], t[\lfloor n/2 \rfloor..j-1])$. Predpostavka $s[i] = t[j]$ implicira tudi $lcs(s[k..i], t[\lfloor n/2 \rfloor..j]) = lcs(s[k..i-1], t[\lfloor n/2 \rfloor..j-1]) + 1$, zato sklepamo $lcs(s[0..i], t[0..j]) = lcs(s[0..k-1], t(0..\lfloor n/2 \rfloor - 1]) + lcs(s[k..i], t[\lfloor n/2 \rfloor..j])$, kar nam glede na definicijo P da, $P[i, j] = k = P[i-1, j-1]$.

Zadnja dva primera obravnavamo na podoben način. □

Spodaj je podana koda algoritma LCS (8) v obliki rekurzivne funkcije, na Tabeli in drevesu 4.6 pa je predstavljeno njegovo delovanje.

Trditev 19. *Algoritem LCS(s, m, t, n) izračuna najdaljše skupno podzaporedje nizov s in t, pripadajočih dolžin m in n.*

Dokaz. Dokažimo algoritem glede na različno dolžino n niza t. Ko je $n = 0$ ali $n = 1$, je trditev enostavno preveriti.

Preverimo sedaj primer, ko je $n > 1$. Če je $m = 0$ ali $m = 1$, samo preverimo, da algoritem res izračuna najdaljše skupno podzaporedje od s in t. Sedaj lahko predpostavimo, da je $m > 1$.

Opazimo, da se v vrsticah $12 - 24$ nadaljuje preračunavanje tabele C_1, ki je bila prvotno izračunana s klicem algoritma LCS-COLUMN v vrstici 8, z nadaljnjo uporabo rekurzivne relacije. Poleg tega opazimo, da je tabela P_1,

Algoritem 8 LCS(s, m, t, n)

1: **if** $m = 1$ **and** $s[0] \in alph(t)$ **then**
2: **return** $s[0]$
3: **else if** $n = 1$ **and** $t[0] \in alph(s)$ **then**
4: **return** $t[0]$
5: **else if** $m = 0$ **or** $m = 1$ **or** $n = 0$ **or** $n = 1$ **then**
6: **return** ε
7: **end if**
8: $C_1 \leftarrow$ LCS-COLUMN$(s, m, t, \lfloor n/2 \rfloor)$
9: **for** $i \leftarrow -1$ **to** $m - 1$ **do**
10: $P_1[i] \leftarrow i + 1$
11: **end for**
12: **for** $j \leftarrow \lfloor n/2 \rfloor$ **to** $n - 1$ **do**
13: $(C_2[-1], P_2[-1]) \leftarrow (0, 0)$
14: **for** $i \leftarrow 0$ **to** $m - 1$ **do**
15: **if** $s[i] = t[j]$ **then**
16: $(C_2[i], P_2[i]) \leftarrow (C_1[i - 1] + 1, P_1[i - 1])$
17: **else if** $C_1[i] > C_2[i - 1]$ **then**
18: $(C_2[i], P_2[i]) \leftarrow (C_1[i], P_1[i])$
19: **else**
20: $(C_2[i], P_2[i]) \leftarrow (C_2[i - 1], P_2[i - 1])$
21: **end if**
22: **end for**
23: $(C_1, P_1) \leftarrow (C_2, P_2)$
24: **end for**
25: $k \leftarrow P_1[m - 1]$
26: $u \leftarrow$ LCS$(s[0..k - 1], k, t[0..\lfloor n/2 \rfloor - 1], \lfloor n/2 \rfloor)$
27: $v \leftarrow$ LCS$(s[k..m - 1], m - k, t[\lfloor n/2 \rfloor..n - 1], n - \lfloor n/2 \rfloor)$
28: **return** $u \cdot v$

ki je implementacija tabele P, izračunana na podlagi rekurzivne formule iz trditve (18), kar je rezultat pravilnega izračuna tabele C_1. Tako dobimo takoj po izvedbi vrstice 25 enakost $k = P_1[m-1] = P[m-1, n-1]$, kar pomeni, da je $lcs(s,t) = lcs(s[0..k-1], t[0..\lfloor n/2 \rfloor - 1]) + lcs(s[k..m-1], t[\lfloor n/2 \rfloor..n-1])$ glede na definicijo P. Rekurzivna klica algoritma v vrsticah 26 in 27 zagotovita najdaljši skupni podzaporedji dveh vhodnih nizov (ki sta bila pravilno izbrana), njuna konkatenacija pa predstavlja najdaljše skupno podzaporedje od s in t.

S tem sta rekurzija in dokaz zaključena. □

Trditev 20. *Algoritem LCS(s, m, t, n) se izvede v času* $\Theta(mn)$ *in prostoru* $\Theta(m)$.

Dokaz. S prvotnim izvajanjem algoritma od 1. do 25. vrstice se ta izvede v času $\Theta(mn)$ (glej Sliko 4.6, zgornji primer).

Ponovno izvajanje teh vrstic s takojšnjim zaporednim klicem vrstic 26 in 27 traja sorazmerno s $k \times \lfloor n/2 \rfloor$ in z $(m-k) \times (n - \lfloor n/2 \rfloor)$, torej globalno gledano $(m \times n)/2$ (glej Sliko 4.6, srednji primer).

Sledi, da je globalni čas izvajanja enak $O(mn)$, saj je $\sum_i (m \times n)/2^i \leq 2(m \times n)$. Zaradi prvega koraka pa drži tudi $\Omega(mn)$, kar nam potrdi prvi del trditve.

Pomnilniški prostor uporablja algoritem LCS za hrambo tabel C_1, C_2, P_1 in P_2, skupaj z nekaterimi spremenljivkami, ki zasedejo konstanten prostor. Skupaj je to $O(m)$ prostora. In ker rekurzivni klici ne zahtevajo hrambe podatkov v tabelah, se lahko njihov prostor ponovno uporabi za nadaljnje izračune. Torej rezultat drži. □

Izrek 2. *Najdaljše skupno podzaporedje med dvema nizoma, dolžin* m *in* n*, je mogoče izračunati v času* $O(mn)$ *in prostoru* $O(\min\{m, n\})$.

Dokaz. Izrek je neposredna posledica trditev (19) in (20), kjer za s izberemo krajšega izmed vhodnih nizov algoritma LCS. □

4.5 Poravnave z vrzelmi

Vrzel (angl. gap) je zaporedje presledkov oz. lukenj v poravnavi. Pri računanju poravnav genetskih zaporedij se pokaže, da je včasih bolj zaželeno globalno kaznovati daljše vrzeli s pomočjo funkcije, ki se povečuje počasneje kot vsota kazni operacij brisanja in vstavljanja posameznih znakov.

V tem kontekstu je minimalni strošek zaporedja operacij urejanja enak razdalji oz. metriki pod enakimi pogoji kot pri trditvi (2), predvsem zaradi tega, ker se upošteva simetričnost med operacijama brisanja in vstavljanja. Predstavimo funkcijo

$$gap : \mathbb{N} \to \mathbb{R},$$

katere vrednost $gap(k)$ predstavlja strošek vrzeli, dolžine k.

Algoritem za izračun poravnave z vrzeljo lahko dobimo dokaj enostavno s predelavo algoritma GENERIC-DP (1) za izračun utežene Levenshteinove razdalje.

V tem primeru za izračun optimalne poravnave potrebujemo tri tabele: D, I in T, velikosti $(m+1) \times (n+1)$. Vrednost $D[i,j]$ pomeni strošek optimalne poravnave nizov $s[0..i]$ in $t[0..j]$, ki se konča z brisanji znakov niza s (vrzel v nizu t). Vrednost $I[i,j]$ predstavlja strošek optimalne poravnave nizov $s[0..i]$ in $t[0..j]$, ki se konča z vstavljanji znakov niza t (vrzel v nizu s). Vrednost $T[i,j]$ nam na koncu da strošek optimalne poravnave nizov $s[0..i]$ in $t[0..j]$. Tabele so med sabo povezane z rekurzivno formulo, ki je predstavljena v naslednji trditvi.

Trditev 21. *Strošek $T[i,j]$ optimalne poravnave nizov $s[0..i]$ in $t[0..j]$ je podan z naslednjo rekurzivno formulo:*

$$\begin{aligned} D[-1,-1] &= D[i,-1] = D[-1,j] = \infty, \\ I[-1,-1] &= \ I[i,-1] = \ I[-1,j] = \infty, \end{aligned}$$

in

$$\begin{aligned} T[-1,-1] &= 0, \\ T[i,-1] &= gap(i+1), \\ T[-1,j] &= gap(j+1), \\ D[i,j] &= \min\{T[l,j] + gap(i-l) : l = 0,1,\ldots,i-1\}, \\ I[i,j] &= \min\{T[i,k] + gap(j-k) : k = 0,1,\ldots,j-1\}, \\ T[i,j] &= \min\{T[i-1,j-1] + Sub(s[i],t[j]), D[i,j], I[i,j]\}, \end{aligned}$$

za $i = 0,1,\ldots,m-1$ *in* $j = 0,1,\ldots,n-1$.

Dokaz. Predlog lahko dokažemo s podobnimi argumenti kot pri trditvi (10). Obravnavati je potrebno tri primere, saj se optimalna poravnava nizov $s[0..i]$ in $t[0..j]$ lahko konča samo na tri različne načine: ali z zamenjavo $t[j]$ s $s[i]$; ali z brisanjem l znakov na koncu s; ali z vstavljanjem k znakov na koncu t, kjer je $0 \leq l < i$ in $0 \leq k < j$. □

Če funkcija *gap* nima nobenih omejitev, potem lahko izračun optimalne poravnave nizov s in t izvedemo v času $O(mn(m+n))$. Po drugi strani pa lahko pokažemo, da je problem rešljiv v času $O(mn)$, če je funkcija *gap* linearna funkcija oblike:

$$gap(k) = g + h \times (k-1),$$

kjer sta g in h dve pozitivni celi števili. Takšen tip funkcije kaznuje začetek vrzeli s količino g in nato različno kaznuje še razširitev vrzeli s količino h. Ponavadi se v praksi izbereta konstanti tako, da velja $h < g$. Rekurzivna formula iz trditve (21) se tako spremeni:

$$\begin{aligned} D[i,j] &= \min\{D[i-1,j] + h, T[i-1,j] + g\}, \\ I[i,j] &= \min\{I[i,j-1] + h, T[i,j-1] + g\}, \\ T[i,j] &= \min\{T[i-1,j-1] + Sub(s[i],t[j]), D[i,j], I[i,j]\}, \end{aligned} \tag{4.11}$$

za $i = 0,1,\ldots,m-1$ in $j = 0,1,\ldots,n-1$. Poleg tega določimo

$$\begin{aligned} D[-1,-1] = D[i,-1] = D[-1,j] &= \infty, \\ I[-1,-1] = \ I[i,-1] = \ I[-1,j] &= \infty, \end{aligned} \tag{4.12}$$

za $i = 0, 1, \ldots, m-1$ in $j = 0, 1, \ldots, n-1$, in

$$\begin{aligned} T[-1,-1] &= 0, \\ T[0,-1] &= g, \\ T[-1,0] &= g, \\ T[i,-1] &= T[i-1,-1] + h, \\ T[-1,j] &= T[-1,j-1] + h, \end{aligned} \tag{4.13}$$

za $i = 1, 2, \ldots, m-1$ in $j = 1, 2, \ldots, n-1$.

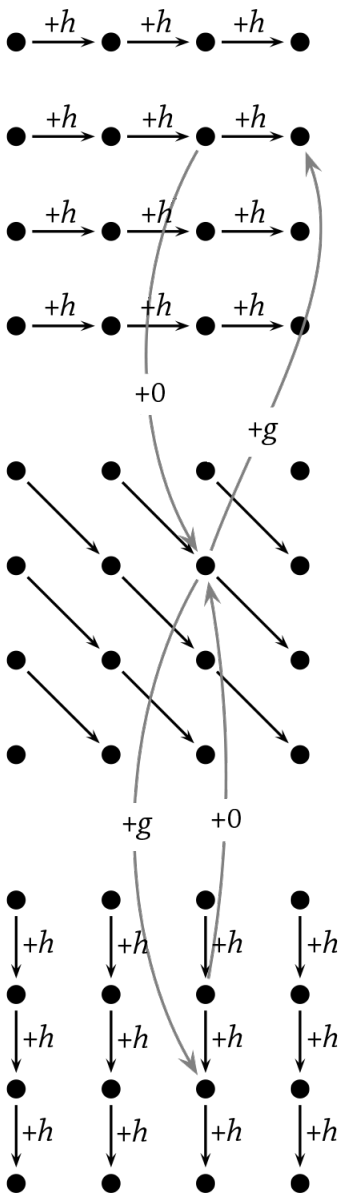
Na Sliki 4.7 je prikazano, kako lahko z grafom urejanja predstavimo poravnave z vrzelmi pod zgornjimi rekurzivnimi pogoji (4.11), (4.12), (4.13). V tem primeru je graf urejanja sestavljen iz treh nivojev. Spodnji nivo grafa je sestavljen iz samih navpičnih povezav z utežmi $+h$ (zaporedne operacije brisanja), srednji nivo sestavljajo diagonalne povezave z utežmi $Sub(s[i], t[j])$, zgornji nivo pa vodoravne povezave z utežmi $+h$ (zaporedne operacije vstavljanja). Spodnji nivo določa vrzeli v nizu t, srednji predstavlja zamenjave in ujemanja ter zgornji nivo vrzeli v nizu s. V tem grafu potekajo tudi povezave iz vsakega vozlišča $(i,j)_{srednji}$ v srednjem nivoju v vozlišča $(i+1,j)_{spodnji}$ v spodnjem nivoju in $(i,j+1)_{zgornji}$ v zgornjem nivoju. Te povezave predstavljajo začetek vrzeli in imajo uteži vrednosti $+g$. Povezave, ki potekajo od $(i,j)_{spodnji}$ in $(i,j)_{zgornji}$ do $(i,j)_{srednji}$, pa predstavljajo konec vrzeli in imajo utež enako 0.

Algoritem GAP (9) uporablja zgornjo rekurzivno formulo (4.11), (4.12), (4.13). Primer izvršitve tega algoritma je predstavljen s Tabelo 4.7.

V algoritmu uporabljene tabele D, I in T se lahko kot v prejšnjem razdelku (najdaljše skupno podzaporedje) predelajo tako, da zasedejo samo linearen prostor.

Sledeča trditev povzema rezultate tega razdelka.

Trditev 22. *Optimalna poravnava nizov, dolžine m in n, kjer je strošek definiran z linearno funkcijo gap, se lahko izračuna v času $O(mn)$ in prostoru $O(\min\{m,n\})$.*



Slika 4.7: Graf urejanja v treh nivojih za prikaz poravnave z vrzeljo, določene z linearno funkcijo *gap*. Vsako vozlišče (i, j) v srednjem nivoju ima eno povezavo, ki gre v zgornji nivo, in eno povezavo, ki gre v spodnji nivo, ter dve povezavi, ki prideta iz zgornjega in spodnjega nivoja. Vir slike: [JP04].

Algoritem 9 GAP(s, m, t, n)

for $i \leftarrow -1$ **to** $m-1$ **do**
  $(D[i,-1], I[i,-1]) \leftarrow (\infty, \infty)$
end for
$T[-1,-1] \leftarrow 0$
$T[0,-1] \leftarrow g$
for $i \leftarrow 1$ **to** $m-1$ **do**
  $T[i,-1] \leftarrow T[i-1,-1] + h$
end for
$T[-1,0] \leftarrow g$
for $j \leftarrow 1$ **to** $n-1$ **do**
  $T[-1,j] \leftarrow T[-1,j-1) + h$
end for
for $j \leftarrow 0$ **to** $n-1$ **do**
  $(D[-1,j], I[-1,j]) \leftarrow (\infty, \infty)$
  for $i \leftarrow 0$ **to** $m-1$ **do**
    $D[i,j] \leftarrow \min\{D[i-1,j] + h, T[i-1,j] + g\}$
    $I[i,j] \leftarrow \min\{I[i,j-1] + h, T[i,j-1] + g\}$
    $t \leftarrow T[i-1,j-1] + Sub(s[i], t[j])$
    $T[i,j] \leftarrow \min\{t, D[i,j], I[i,j]\}$
  end for
end for
return $T[m-1,n-1]$

D	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	∞	∞	∞	∞	∞	∞	∞
0	T	∞	6	7	8	9	10	11
1	A	∞	6	6	9	10	11	10
2	C	∞	7	7	9	11	12	11
3	T	∞	8	8	9	9	12	12
4	C	∞	9	9	10	10	12	12

I	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	∞	∞	∞	∞	∞	∞	∞
0	T	∞	6	6	6	7	8	9
1	A	∞	7	8	9	9	10	11
2	C	∞	8	9	10	9	9	10
3	T	∞	9	10	10	11	12	12
4	C	∞	10	11	12	10	11	12

T	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	0	$\leftarrow 3$	4	5	6	7	8
0	T	3	3	$\nwarrow 3$	6	7	8	7
1	A	4	6	6	$\nwarrow 6$	9	10	10
2	C	5	7	7	6	$\nwarrow 6$	$\leftarrow 9$	10
3	T	6	8	7	9	9	$\nwarrow 9$	$\nwarrow 9$
4	C	7	9	9	7	9	11	$\nwarrow\uparrow 12$

Tabela 4.7: Prikaz izračuna tabel D, I in T z algoritmom GAP na primeru nizov TACTC in GTCCGT. Uporabimo vrednosti $g = 3, h = 1, Sub(a, a) = 0$ in $Sub(a, b) = 3$ za $a, b \in \Sigma$, kjer $a \neq b$. Dobimo rezultat $T[4, 5] = 12$.

4.6 Lokalna poravnava

Pogosto je namesto obravnave globalne poravnave nizov s in t bolj ustrezno določiti najboljšo poravnavo podnizov $s[l..i]$ in $t[k..j]$, čemur rečemo lokalna poravnava. Slika 4.8 predstavlja primerjavo teh dveh pristopov. V primeru lokalne poravnave uporaba pojma razdalje ni najbolj primerna. Ko namreč želimo minimizirati razdaljo, so podnizi, ki vodijo do najmanjših vrednosti, tisti podnizi, ki se pojavljajo vzporedno v obeh nizih s in t, ter lahko v tem primeru obsegajo le par znakov. Namesto tega raje uporabimo pojem podobnost med nizi, kjer se enakosti znakov vrednotijo pozitivno, neenakosti, vstavljanja in brisanja pa negativno. Iskanje podobnih podnizov je torej sestavljeno iz maksimizacije količine, ki meri podobnost med nizoma.

Problem 6 (Lokalna optimalna poravnava nizov)**.**
Naloga: Končna abeceda Σ, niza s in t iz Σ^* in funkcija ocene $score(a, b)$, $a, b \in \Sigma$.
Dopustna rešitev: Lokalna poravnava nizov s in t; tj. globalna poravnava $\alpha = \big(\tilde{s}[l..i], \tilde{t}[k..j]\,\big)$ podnizov $s[l..i]$ in $t[k..j]$.
Mera/ciljna funkcija: Ocena lokalne poravnave $score(\alpha)$ nizov s, t, ki je enaka vsoti ocen posameznih poravnanih parov $score(a, b)$; tj. ocena globalne poravnave podnizov $score\big(\tilde{s}[l..i], \tilde{t}[k..j]\,\big)$.
Cilj: Maksimizacija.

4.6.1 Podobnost

Za merjenje podobnosti med dvema nizoma s in t uvedemo *funkcijo ocene* (angl. score function). Funkcija, označena Sub_s, meri stopnjo podobnosti med dvema znakoma v abecedi. Večja kot je vrednost $Sub_s(a, b)$, bolj sta si znaka med sabo podobna. Predpostavimo, da je funkcija:

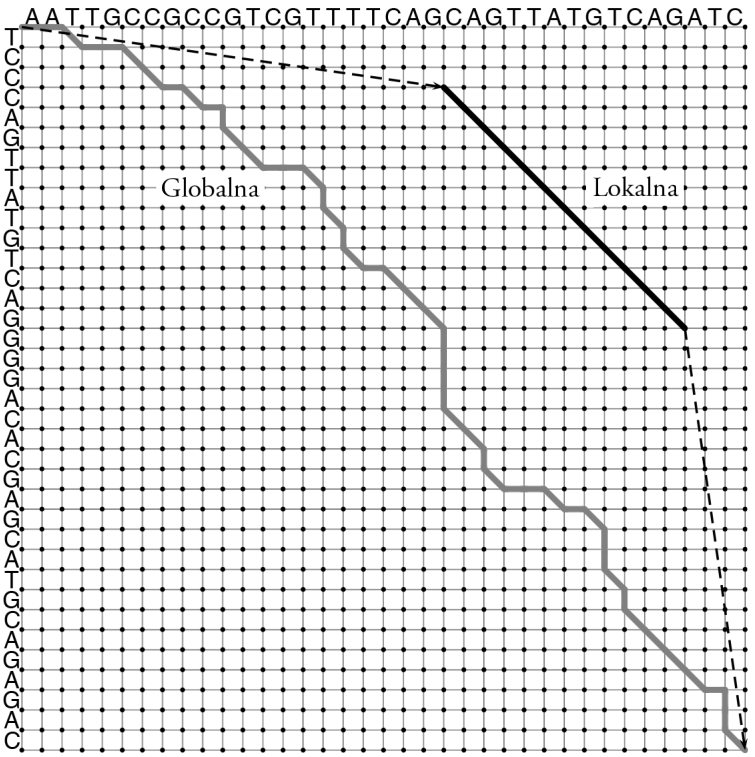
$$Sub_s(a, a) > 0$$

za $a \in \Sigma$ in

$$Sub_s(a, b) < 0$$

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || |  | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C

                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                    ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```



Slika 4.8: Primerjava med lokalno in globalno poravnavo nizov. Vir slike: [JP04].

za $a, b \in \Sigma$, kjer $a \neq b$. Funkcija Sub_s je simetrična, vendar ni razdalja, saj ne zadovoljuje pogojev nenegativnosti, ločenosti in trikotniške neenakosti. Poleg tega lahko različnim enakostim med znaki dodelimo različne ocene: lahko imamo $Sub_s(a, a) \neq Sub_s(b, b)$. To nam dovoli boljši nadzor nad enakostmi, ki so bolj zaželene. Vstavljanja in brisanja morajo biti negativno ocenjena (njihove vrednosti so cela števila):

$$Ins_s(a) < 0$$

in

$$Del_s(a) < 0$$

za $a \in \Sigma$.

Definicija 46. Podobnost $sim(s, t)$ *(angl. similarity) med nizoma* s *in* t *tako definiramo:*

$$sim(s, t) = \max_{\sigma \in \mathcal{S}(s,t)} score(\sigma),$$

kjer je $\mathcal{S}(s, t)$ *množica zaporedij operacij urejanja, ki preoblikujejo* s *v* t*;* $score(\sigma)$ *pa predstavlja vsoto posameznih ocen operacij urejanja iz* σ*.*

Predstavimo naslednjo trditev, ki vzpostavi razmerje med pojmoma razdalje in podobnosti.

Trditev 23. *Dane so: razdalja med znaki Sub, dve funkciji nad znaki Ins in Del ter konstanti* g *in* l*. Sistem točkovanja definiramo na naslednji način:*

$$Sub_s(a, b) = l - Sub(a, b)$$

in

$$Ins_s(a) = Del_s(a) = -g + \frac{l}{2}$$

za vse znake $a, b \in \Sigma$*. Potem imamo:*

$$ED(u, v) + sim(u, v) = \frac{l}{2}(|u| + |v|)$$

za vsak niz $u, v \in \Sigma^*$*.*

Dokaz trditve (23) lahko najdemo v [SM97].

4.6.2 Izračun optimalne lokalne poravnave

Optimalna lokalna poravnava nizov s in t je par nizov (u, v), za katere velja $u \preceq_{fakt} s$, $v \preceq_{fakt} t$ in $sim(u, v)$ je maksimum. Za postopek izračuna, podobnega kot pri posplošeni Levenshteinovi razdalji, uporabimo tabelo S, definirano za $i = -1, 0, \ldots, m-1$ in $j = -1, 0, \ldots, n-1$, kjer je $S[i, j]$ maksimalna podobnost med pripono od $s[0..i]$ in pripono od $t[0..j]$. Ali tudi:

$$S[i, j] = \max\{sim(s[l..i], t[k..j]) : 0 \leq l \leq i \text{ in } 0 \leq k \leq j\} \cup \{0\}$$

je ocena lokalne poravnave v $[i, j]$. Optimalno lokalno poravnavo se potem izračuna z vzvratnim pregledom tabele od maksimalne vrednosti proti vozlišču $[-1, -1]$.

Trditev 24. *Tabela S izpolnjuje rekurzivno relacijo:*

$$S[i, j] = \max \begin{cases} 0, \\ S[i-1, j-1] + Sub_s(s[i], t[j]), \\ S[i-1, j] + Del_s(s[i]), \\ S[i, j-1] + Ins_s(t[j]), \end{cases}$$

in

$$S[-1, -1] = S[i, -1] = S[-1, j] = 0$$

za $i = 0, 1, \ldots, m-1$ in $j = 0, 1, \ldots, n-1$.

Dokaz. Dokaz je analogen tistemu iz trditve (10). □

Sledeči algoritem LOCAL-ALIGNMENT (10) predstavlja izvedbo rekurzivne formule prejšnje trditve (24).

Trditev 25. *Algoritem LOCAL-ALIGNMENT izračuna oceno vseh lokalnih poravnav nizov s in t.*

Dokaz. Algoritem uporablja rekurzivno relacijo iz trditve (24). □

Da najdemo optimalno lokalno poravnavo, moramo najprej locirati največjo vrednost v tabeli S. Nato od te pozicije poiščemo pot nazaj navzgor

Algoritem 10 LOCAL-ALIGNMENT(s, m, t, n)

1: **for** $i \leftarrow -1$ **to** $m-1$ **do**
2: $\quad S[i,-1] \leftarrow 0$
3: **end for**
4: **for** $j \leftarrow 0$ **to** $n-1$ **do**
5: $\quad S[-1,j] \leftarrow 0$
6: $\quad$ **for** $i \leftarrow 0$ **to** $m-1$ **do**
7: $$S[i,j] \leftarrow \max \begin{cases} 0 \\ S[i-1,j-1] + Sub_s(s[i],t[j]) \\ S[i-1,j] + Del_s(s[i]) \\ S[i,j-1] + Ins_s(t[j]) \end{cases}$$
8: $\quad$ **end for**
9: **end for**
10: **return** S

v smeri S[-1,-1] in s tem pregledamo tabelo tako kot pri algoritmu ONE-ALIGNMENT (2). Pregled ustavimo, ko dosežemo vrednost nič. Primer je predstavljen v Tabeli 4.8.

S	j	-1	0	1	2	3	4	5
i		$t[j]$	G	T	C	C	G	T
-1	$s[i]$	0	0	0	0	0	0	0
0	T	0	0	1	0	0	0	1
1	A	0	0	0	0	0	0	0
2	C	0	0	0	1	1	0	0
3	T	0	0	↖1	0	0	0	1
4	C	0	0	0	↖2	1	0	0

Tabela 4.8: Izračun optimalne lokalne poravnave med nizoma $s =$ TACTC in $t =$ GTCCGT, ko so $Sub_s(a, a) = 1$, $Sub(a, b) = -3$ in $Del_s(a) = Ins_s(a) = -1$ za $a, b \in \Sigma$, $a \neq b$. Predstavljena je tabela S stroškov vseh lokalnih poravnav dobljena z algoritmom LOCAL-ALIGNMENT. V njej je označena maksimalna pot, ki se konča na poziciji z največjo vrednostjo $[4, 2]$.

Poglavje 5

Sorodni problemi

Predstavimo še probleme, ki so povezani z merami podobnosti nizov ali pa predstavljajo njihovo nadgradnjo [Ste94], [Nav01].

5.1 Najdaljše skupno podzaporedje N nizov

Posplošitev $Lcs(s,t)$ problema je N-$Lcs(S)$ problem, kjer je potrebno poiskati najdaljše podzaporedje skupno N nizom iz množice S.

Problem 7 (Najdaljše skupno podzaporedje N nizov)**.**
Naloga: Končna abeceda Σ, končna množica N nizov iz Σ^*: $S = \{s_1, \ldots, s_N\}$.
Dopustna rešitev: Niz $w \in \Sigma^*$ tako, da je w podzaporedje vsakega $s_i \in S$, $i = 1, \ldots, N$; tj. w dobimo z odvzemom znakov iz vsakega s_i.
Mera/ciljna funkcija: Dolžina podzaporedja, $|w|$.
Cilj: Maksimizacija.

Po tej definiciji se osnovni $Lcs(s,t)$ problem (5) imenuje 2-$Lcs(s,t)$ problem.

Problem določitve dolžine Najdaljšega skupnega podzaporedja N nizov N-$lcs(S)$ je NP-težek za abecede velikosti 2 in več. Nadgradnja metode dinamičnega programiranja s pomočjo N-dimenzionalne tabele nam tako da $O(Nn^N)$ časovno in prostorsko zahtevnost za nize dolžine n.

5.2 Najkrajše skupno super-zaporedje in super-niz

Problem NAJKRAJŠEGA SUPER-ZAPOREDJA MNOŽICE N NIZOV (angl. shortest common supersequence of N strings) predstavlja iskanje zaporedja minimalne dolžine N-$Scs(S)$, tako da je to zaporedje skupno super-zaporedje od vseh N nizov iz množice S.

Problem 8 (NAJKRAJŠE SKUPNO SUPER-ZAPOREDJE N NIZOV)**.**
Naloga: Končna abeceda Σ, končna množica N nizov iz Σ^*: $S = \{s_1, \ldots, s_N\}$.
Dopustna rešitev: Niz $w \in \Sigma^*$ tako, da je vsak $s_i \in S$, $i = 1, \ldots, N$, podzaporedje od w; tj. s_i dobimo z odvzemom znakov iz w.
Mera/ciljna funkcija: Dolžina super-zaporedja, $|w|$.
Cilj: Minimizacija.

Problem N-$scs(S)$ je za abecede velikosti ≥ 2 NP-težek.

Podoben je problem NAJKRAJŠEGA SKUPNEGA SUPER-NIZA (angl. shortest common superstring), kjer je potrebno najti niz minimalne dolžine w, tako da je vsak $s_i \in S$, $i = 1, \ldots, N$, podniz od w. Tudi ta problem je NP-težek. Problem je v praksi zelo pomemben na področju molekularne biologije. Zaradi omejitev algoritmov za obdelavo nizov moramo dolge molekule (npr. človeški genom) pred analizo narezati na krajše odseke. Po obdelavi krajših vzorcev moramo odseke sestaviti nazaj v super-niz, ki predstavlja celotno molekulo. Pri tem se uporablja metoda NAJKRAJŠEGA SKUPNEGA SUPER-NIZA.

5.3 Najkrajše skupno ne-podzaporedje N nizov

Problem N-$Scns(S)$ (angl. shortest common non-subsequence of N strings) se uporablja na področju strojništva. Gre za iskanje najkrajšega niza nad abecedo Σ, ki ni podzaporedje nobenega izmed N nizov iz množice S.

Problem 9 (Najkrajše skupno ne-podzaporedje N nizov)**.**
Naloga: Končna abeceda Σ, končna množica N nizov iz Σ^*: $S = \{s_1, \ldots, s_N\}$.
Dopustna rešitev: Niz $w \in \Sigma^*$ tako, da w ni podzaporedje nobenega $s_i \in S$, $i = 1, \ldots, N$.
Mera/ciljna funkcija: Dolžina ne-podzaporedja, $|w|$.
Cilj: Minimizacija.

Odločitvena variacija tega problema (določitev ali je N-$scns(S)$ manjši od določene vrednosti) je NP-težka za abecede velikosti ≥ 2.

5.4 Najtežje skupno podzaporedje

$Hcs(s,t)$ je podzaporedje, skupno s in t, kjer mora biti vsota uteži posameznih znakov maksimalna. Uteži so določene s funkcijo $w(a)$, ki je lahko odvisna tako od znakov samih kot tudi od njihovih položajev v prvotnem nizu.

Problem 10 (Najtežje skupno podzaporedje)**.**
Naloga: Končna abeceda Σ, niza s in t iz Σ^*, funkcija uteži $w(a), a \in \Sigma$.
Dopustna rešitev: Niz $w \in \Sigma^*$ tako, da je w skupno podzaporedje od s in t; tj. w dobimo z odvzemom znakov iz niza s ali niza t.
Mera/ciljna funkcija: Teža podzaporedja, ki je enaka vsoti posameznih uteži znakov $w(a)$.
Cilj: Maksimizacija.

Najtežje skupno podzaporedje $hcs(s,t)$ (angl. heaviest common subsequence) se je včasih uporabljalo za osveževanje zaslonov s katodno cevjo. Operacije, potrebne za preoblikovanje vrstice iz enega zaslona v drugega, so izpeljane iz $lcs(s,t)$ dveh zaslonov. Z namenom minimizacije motenj zaslona je boljše dati prednost čim bolj usklajenim parom ujemajočih vrstic, zato se zraven uporabi funkcija uteži w, ki izboljša njegovo delovanje. Za izračun $hcs(s,t)$ se lahko predela algoritem za izračun $lcs(s,t)$ na osnovi dinamičnega programiranja.

5.5 Približno iskanje vzorca v besedilu

Nadgradnja približnega primerjanja nizov je iskanje vseh približnih ujemanj vzorca v besedilu (angl. approximate string matching, tudi fuzzy string searching). Torej iščemo vsa podzaporedja oz. njihove pozicije v besedilu, ki so za največ k neujemanj oddaljena od vzorca ($ED(p, t[i..m-n+1]) \leq k$).

Problem 11 (Približno iskanje vzorca v besedilu)**.**
Naloga: Končna abeceda Σ, vzorec $p \in \Sigma^*$, dolžine n, besedilo $t \in \Sigma^*$, dolžine m, množica operacij urejanja $\mathbb{B}$, stroškovna funkcija δ in konstanta k.
Dopustna rešitev: Za vsako pozicijo i v nizu t, $1 \leq i \leq m-n+1$, določitev sledi urejanja σ, ki predstavlja zaporedje operacij urejanja iz $\mathbb{B}$, ki pretvorijo $p[1..n]$ v $t[i..i+n-1]$.
Mera/ciljna funkcija:[1] *Za vsako pozicijo i, $1 \leq i \leq m-n+1$, določitev stroška sledi urejanja $cost(\sigma)$ med $t[i..i+n-1]$ in $p[1..n]$, ki je vsota posameznih stroškov $\delta(u \to v)$, $u \to v \in \mathbb{B}$.*
Cilj: Izbor tistih pozicij i, za katere je strošek sledi urejanja $cost(\sigma)$ največ k.

Algoritem na osnovi dinamičnega programiranja za Iskanje približnega vzorca v besedilu se izvede v času $O(nm)$, poznamo pa tudi verzijo, ki je v najslabšem primeru $O(km)$.

Metode iskanja nizov lahko razvrstimo na aktivne (on-line) in neaktivne (off-line) metode. On-line iskalne metode vključujejo algoritme za iskanje vzorcev v vnaprej neobdelanih tekstovnih nizih. Ker je čas izvajanja on-line algoritmov sorazmeren z velikostjo besedila, je ta pristop precej neučinkovit. Zaradi tega se v praksi pogosto uporabljajo off-line algoritmi, ki temeljijo na predhodni obdelavi besedila, kot je na primer indeksiranje.

[1] Odvisno od množice $\mathbb{B}$ in stroškovne funkcije δ je mera lahko: Levenshteinova razdalja, Damerau-Levenshteinova razdalja, Hammingova razdalja, Indel razdalja, posplošena Levenshteinova razdalja ...

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo predstavili mere podobnosti oz. različnosti nizov in s tem povezan problem približnega primerjanja nizov. Ker je veliko podatkov danes še vedno predstavljenih v obliki nizov in ker ti podatki prihajajo iz zelo različnih področij uporabe, njihova uporabnost pa je zelo pomembna, pomeni, da je bilo naše področje proučevanja zelo obširno. To je tudi razlog, da smo se pri bolj podrobni analizi mer podobnosti omejili samo na mere na osnovi operacij urejanja nizov.

Pomemben del naloge je bil postaviti trdne teoretične temelje ter tako povezati praktični vidik mer podobnosti z njihovo matematično osnovo. Pomembno je bilo najprej razjasniti pojme podobnosti, različnosti in razdalje, kar smo črpali iz teorije merjenja in razvrščanja v skupine. V literaturi s področja računalništva je ta vidik teoretične predstavitve mer zanemarjen, saj česa več od definicije razdalje ne najdeš pogosto. Problematično pa je tudi poimenovanje istih mer z različnimi izrazi. Tako sta na primer definiciji razdalje urejanja in Levenshteinove razdalje na različnih mestih predstavljeni z različnimi koncepti, nekateri avtorji pa ju celo enačijo. Mi smo definirali razdaljo urejanja kot splošen pojem najmanjšega števila operacij urejanja, potrebnega za preoblikovanje prvega niza v drugega. Levenshteinovo razdaljo pa smo poimenovali kot poseben primer razdalje urejanja, kjer so dovoljene operacije vstavljanja, brisanja in zamenjave, katerih strošek je enak ena.

Ko smo razjasnili zmedo[1] glede poimenovanja mer podobnosti, je temu sledila še predstavitev načinov vizualizacije razdalje urejanja. Ti so namreč v veliko pomoč pri razumevanju podrobnejšega opisa postopkov izračunov in izvedb algoritmov v nadaljevanju. Poleg osnovnih algoritmov po principu dinamičnega programiranja za izračun posplošene Levenshteinove razdalje in pripadajoče poravnave ter najdaljšega skupnega podzaporedja smo dodali še izpeljave, ki predstavljajo prostorsko izboljšavo. Nato smo obravnavali še dva algoritma za izračun poravnav, pri prvem je bil strošek operacij urejanja definiran z linearno funkcijo *gap*, v drugem primeru pa je šlo za optimalno lokalno poravnavo.

Skozi celotno nalogo smo z namenom boljšega razumevanja obdelovali isti primer primerjave nizov.

Sedaj lahko rečemo, da poznamo razliko med globalno in lokalno poravnavo nizov, med Levenshteinovo in Damerau-Levenshteinovo razdaljo ter med omejeno in neomejeno razdaljo urejanja. Vemo, kakšne omejitve ima Hammingova razdalja in zakaj smo v nalogi predstavili tudi najdaljše skupno podzaporedje nizov.

Področje raziskovanja mer podobnosti nizov bi lahko razširili na hevristične in aproksimativne razdalje. Lahko bi obdelali tudi področje mer podobnosti dreves, grafov ... Lahko bi predstavili izboljšane verzije algoritmov, kot so na primer paralelni algoritmi ter ostale novejše časovne in prostorske različice. Eno izmed zanimivih področij je vložitev mer podobnosti v druge prostore (npr. vektorskega), kar bi nam lahko odprlo nove prostore oz. svetove.

[1] Dopuščam možnost, da je bila morda zmeda samo v moji glavi.

Literatura

[Bat85] Vladimir Batagelj. *Razvrščanje v skupine: nehierarhični postopki.* magistrsko delo, 1985.

[Boy11] Leonid Boytsov. Indexing methods for approximate dictionary searching. *Journal of Experimental Algorithmics*, page 93, 2011.

[Bre98] Matevž Bren. *Mere različnosti.* Disertacija, Univerza v Ljubljani, 1998.

[CHL01] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings.* Cambridge University Press, 2001.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* MIT Press, third edition, 2009.

[CZ08] Kun-Mao Chao and Louxin Zhang. *Sequence Comparison. Theory and Methods.* 2008.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[Hir75] D S Hirschberg. Algorithm for Computing Maximal Common Subsequences. 18(6):2–4, 1975.

[HUM01] John E. Hopcroft, Jeffrey D. Ullman, and Rajeev Motwani. Introduction to Automata Theory, Languages, and Computation, 2001.

[JP04] Neil C. Jones and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*, volume 101. MIT press, 2004.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals, 1966.

[Nav01] Gonzalo Navarro. A Guided Tour to Approximate String Matching. *Computer*, 33(1):31–88, 2001.

[SM97] Joao Setubal and Joao Meidanis. *Introduction To Computational Molecular Biology*. 1997.

[Ste94] Graham A Stephen. *String searching algorithms*. 1994.

[Str15] String (computer Science). https://en.wikipedia.org/wiki/String_(computer_science), 2015.

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985.

[Vér88] Jean Véronis. Computerized correction of phonographic errors. *Computers and the Humanities*, 22(1):43–56, 1988.

[WF74] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *Journal of the ACM*, 21(1):168–173, 1974.

[WL75] Robert A. Wagner and Roy Lowrance. An Extension of the String-to-String Correction Problem. *Journal of the ACM*, 22(2):177–183, 1975.

Tabele

Slike

Algoritmi